conference

proceedings

The Eleventh Systems Administration Conference (LISA '97) Proceedings

San Diego, California
October 26–31, 1997

Co-sponsored by **The USENIX Association and SAGE, the System Administrators Guild**

USENIX®
The Advanced Computing Systems Association

SAGE
THE SYSTEM ADMINISTRATORS GUILD

USENIX Eleventh Systems Administration Conference (LISA '97) Proceedings *San Diego, California October 1997*

For additional copies of these proceedings contact

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Telephone: 510-528-8649

The price is $30 for members and $38 for nonmembers.

Outside the U.S.A. and Canada, please add
$12 per copy for postage (via air printed matter).

Past USENIX Large Installation Systems Administration Workshop and Conference Proceedings (price: member/nonmember)

Large Installation Systems Admin. I Workshop	1987	Phildelphia, PA	$4/$4
Large Installation Systems Admin. II Workshop	1988	Monterey, CA	$8/$8
Large Installation Systems Admin. III Workshop	1989	Austin, TX	$13/$13
Large Installation Systems Admin. IV Conference	1990	Colorado Spgs, CO	$15/$18
Large Installation Systems Admin. V Conference	1991	San Diego, CA	$20/$23
Systems Administration VI Conference	1992	Long Beach, CA	$23/$30
Systems Administration VII Conference	1993	Monterey, CA	$25/$33
Systems Administration VIII Conference	1994	San Diego, CA	$22/$29
Systems Administration IX Conference	1995	Monterey, CA	$30/$38
Systems Administration X Conference	1996	Chicago, IL	$30/$38

Outside the U.S.A. and Canada, please add $12
per copy for postage (via air printed matter).

ISBN 1-880446-90-1

♻ Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

Proceedings of the Eleventh Systems Administration Conference (LISA XI)

October 26-31, 1997
San Diego, CA, USA

TABLE OF CONTENTS

Working With PCs

Thursday (9:00am-10:30am) — Chair: Melissa Binde

Inside the Black Box

Thursday (11:00am-12:30pm) — Chair: John Sellens

Net Gains

Thursday (4:00pm-5:00pm) — Chair: Josh Simon

Configuration Management

Friday (9:00am-10:30am) — Chair: Paul Anderson

Mail

Friday (11:00-12:30 pm) — Chair: Bill LeFebvre

ACKNOWLEDGMENTS

PROGRAM CO-CHAIRS
Hal Pomeranz, *Deer Run Associates*
Celeste Stokely, *Stokely Consulting*

PROGRAM COMMITTEE
Paul Anderson, *University of Edinburgh*
Melissa Binde, *Swarthmore College*
Helen E. Harrison, *SAS Institute, Inc.*
Trent R. Hein, *XOR Network Engineering*
Amy Kreiling, *SAS Institute, Inc.*
William LeFebvre, *Group sys Consulting*
Dinah McNutt, *IT Masters, Inc.*
Adam Moskowitz, *Genome Therapeutics Corp.*
Wendy Nather, *Swiss Bank Warburg*
John Sellens, *UUNET Canada*
Josh Simon, *Paranet*

INVITED TALKS COORDINATORS
Rik Farrow, *Internet Security Consulting*
Pat Wilson, *Dartmouth College*

WORK-IN-PROGRESS COORDINATOR
Amy Kreiling, *SAS Institute, Inc.*

GURU-IS-IN COORDINATOR
Lee Damon, *Qualcomm*

POSTER SESSION
John Posey, *Scenario Software Systems, Inc*

TERMINAL ROOM COORDINATOR
Lynda McGinley, *Net Daemons Associates, Inc.*

PROCEEDINGS PRODUCTION
Rob Kolstad, *Berkeley Software Design, Inc.*
Data Reproductions

USENIX MEETING PLANNER
Judith F. DesHarnais, *USENIX Association*

USENIX EXECUTIVE DIRECTOR
Ellie Young, *USENIX Association*

USENIX SUPPORT STAFF
Eileen Curtis, *USENIX Association*
Diane DeMartini, *USENIX Association*
Jackson Dodd, *USENIX Association*
Julie Keiser, *USENIX Association*
Toni Veglia, *USENIX Association*
Lana Weeden, *USENIX Association*

USENIX PUBLICATIONS DIRECTOR
Eileen Cohen, *USENIX Association*

USENIX MARKETING DIRECTOR
Zanna Knight, *USENIX Association*

USENIX EXHIBITS COORDINATOR
Cynthia Deno, *USENIX Assocation*

PREFACE

LISA is back at the Town and Country again, but don't let our return to this venue make you think that LISA is set in its ways. Building on the strengths of previous conferences, our extended program committee (including the USENIX Staff, IT and Guru coordinators, and our omnipotent typesetter) has put together a program that will make even jaded, long-time LISA attendees sit up and take notice. Let us take moment to review the conference schedule and thank those responsible for LISA'97.

First, Dan Klein, the USENIX tutorial coordinator, has once again pulled in a prime set of tutorials and instructors. There are never-before-seen courses from top-rated instructors and plenty of new material. We hope you get an opportunity to sample Dan's program.

The book you are holding in your hand and the refereed paper track for the conference are due to the hard work of our Program Committee. Our conference program addresses old favorites (including the last word on configuration management and some new information on the Sendmail front) and growing challenges (PCs and the increasingly complicated world of System Administration). Of course, this program wouldn't exist at all without the submissions from you, our community. We thank you, especially.

Our Program Committee was one of the largest in recent memory, and we would like to thank some members with special responsibilities. Helen Harrison, Melissa Binde, and John Sellens were tasked with bringing in a great keynote for the opening ceremonies – after Wednesday, we think you'll agree they did a first-rate job. Lee Damon has pulled in an amazing Guru-Is-In track, and Pat Wilson and Rik Farrow have further complicated the schedule by pulling together the best Invited Talk track in years. As if there weren't enough to choose from, Nolan Posey and Alva Couch have created something new for LISA: the poster session. Poster sessions are more than a WIP but less than a full paper. Check out the posters, talk one-on-one with the authors, and get some great ideas to take home with you.

Of course, this conference would be happening under a tent in a field without the incredible efforts of the USENIX Office. We probably wouldn't even have the tent if it were not for the efforts of folks like Judy DesHarnais, Zanna Knight, and Ellie Young. When your Program Chairs realized exactly how much we had bitten off, these were the people who picked us up and dusted us off.

Our final thanks go to our able typesetter, Rob Kolstad. Aside from producing the book you now hold in your hands, Rob is our Master of Ceremonies for this year's LISA Quiz Show – a not-to-be-missed event!

Thanks for coming to San Diego, everybody! See you in the hallways!

Hal Pomeranz
Celeste Stokely

AUTHOR INDEX

Implementing a Generalized Tool for Network Monitoring

Marcus J. Ranum, Kent Landfield, Mike Stolarchuk, Mark Sienkiewicz, Andrew Lambeth, and Eric Wall – Network Flight Recorder, Inc.

ABSTRACT

Determining how you were attacked is essential to developing a response or countermeasure. Usually, a system or network manager presented with a successful intrusion has very little information with which to work: a possibly corrupted system log, a firewall log, and perhaps some tcpdump output.

When hackers come up with a new technique for cracking a network, it often takes the security community a while to determine the method being used. In aviation, an aircraft's "black box"[1] is used to analyze the details of a crash. We believe a similar capability is needed for networks. Being able to quickly learn how an attack works will shorten the effective useful lifetime of the attack. Additionally, the recovered attack records may be helpful in tracking or prosecuting the attacker. Since we've developed a general purpose statistics-gathering system, we believe it will be useful for more than just security. For example, a network manager may desire an historical record of the usage growth of certain applications, or details about the breakdown of types of traffic at different times of day. Such records will provide useful information for network managers in diagnosing performance problems or planning growth.

This paper describes an architecture and toolkit for building network traffic analysis and statistical event records: The Network Flight Recorder. The NFR uses a promiscuous packet interface to pass visible traffic into an internally meta-programmed decision engine which routes information about packets and their contents into statistical or logging backends. In addition to packet analysis and collection, the NFR's internal architecture permits network managers to sample interesting portions of network traffic for logging or statistical analysis. The NFR programming language is simple, but powerful enough that you can perform reasonable analysis on traffic before choosing to record it. For example, you might analyze SMTP transactions but only choose to record those relating to a user who is sending spam or abusive E-mail. The analysis language includes a capability for generating alert messages which the rest of the system queues, multiplexes, and delivers. A simplified hyper-query interface allows extensive browsing of the NFR's stored datasets and statistics from any Java-enabled browser. The NFR is currently being deployed at a number of ISPs and commercial sites, and is available for download in source code form from www.nfr.net.[2]

Background and Motivation

In 1990, one of the authors managed a rather chaotic network, including an embryonic firewall, using NNStat as a security tool. NNStat [1] was designed as a statistical analysis system for the NSFnet backbone, not as a security tool, but possessed several attractive properties:

1. It permits accurate and highly condensed summaries of an event on the network.
2. It permits flexible specification of types of events to record.
3. It permits flexible storage of information about the events that are observed.

While NNStat's authors were concerned about, for example, how much RIP traffic was crossing the network, a security conscious network manager could use NNStat to record all RIP traffic emanating from any systems that were not on an "approved list" of routers. Suddenly, NNStat was useful as a crude tool for mapping who and what, as well as for setting an alert to fire when something happened that the network manager believed should not. NNStat, wrapped with a bunch of quick and dirty shell scripts and cron jobs, served well as a poor man's intrusion detection system. Other network managers have implemented similar systems using tcpdump, or more sophisticated special-purpose network watchers like ARPwatch [2], TCPwatch [3], Netman [4], clog [5], Netwatch [6], and Argus [7]. Other intrusion detection burglar alarms have focused on features of the host operating system, such as tcp_wrappers [8], klaxon [9], and tocsin [10]. Many of the monitoring systems implemen-

[1] They are actually Safety Orange.

[2] Use of the NFR software is free for noncommercial and research purposes. A commercial release of the software is being developed.

ted in the past contain features found in NFR. We believe that the new ground the NFR breaks is by making the filtering and analysis process internally programmed, rather than static-coded into the monitoring application.

NFR is intellectually evolved from NNStat, but includes a more generalized and powerful filtering language, as well as the ability to trigger alerts and log complete packet information. A triggering specification lets data be selected from reassembled TCP sessions, providing a powerful capability for usage measurement as well as audit. The authors intend to use NFR as a platform for exploring auditing and logging, while simultaneously providing a freely available, high quality data source for researchers working on intrusion detection.

Overview of the NFR Architecture

The architecture of NFR was designed as a set of components, each tailored to a specific activity. Data is gathered by one or more packet suckers, forwarded to the decision engine for filtering and reassembly, and possibly recorded to a backend for storage or statistical processing. The query interface is kept completely separate from the input data flow to minimize the performance impact of a user's querying the system while it is collecting data.

Packet Suckers

The packet suckers we initially implemented have been based on the libpcap [11] packet capture interface. Libpcap provides a generalized packet capture facility atop a number of operating system-specific network capture interfaces. This freed us from having to deal with a lot of portability issues. We did discover, however, that some of the available packet capture facilities cannot reliably buffer high volumes of bursty traffic. Berkeley packet filter-based packet suckers running on a Pentium-200 were unable to handle even moderate network loads. This was a result of a latency interaction between BPF and our software: we do more processing than a program like tcpdump, and, though our average processing seems to be within the performance envelope of the machine, we can't always process the packet "immediately," as BPF expects. To fix the problem, we increased internal buffer sizes from their default of 32K to 256K, a number more appropriate for the amount of RAM available in modern computers. Since the NFR daemon potentially monitors multiple interfaces, we performed minor modifications to the way blocking and time-outs are performed in BPF. The original BPF time-out is an inter-packet time-out based in the arrival of a packet. If you don't see a packet, you never time out. We modified it to begin the timer with the read() or select() timeout, so we can detect periods of no traffic.

Typical applications using BPF, such as tcpdump, wait in a tight loop while they read packets from the interface. Since we are trying to do additional processing, and potentially additional I/O, we have a closer-to-real-time requirement, which doesn't sit well with a wait-loop model. For high-performance networks, we are examining designing a new packet capture interface based on a memory-mapped device driver. Most of the mechanisms for packet capture require two system calls (or an interrupt and a system call) per packet; we'd like to reduce it to a single semaphore check.

When a packet is collected by a packet sucker, it is passed to the decision engine using a generalized API intended to allow packet suckers to be separate processes from the engine. The libpcap-based packet sucker is compiled into the engine, but we wanted to admit the possibility of multiprocessing packet suckers that might perform their own buffering. Packets read from libpcap include time information, which is preserved with the packet as it is passed through NFR, providing a notion of time to the entire engine.

Decision Engine

Packets are passed into the decision engine, where they are checked against a list of filters for evaluation. Filters are written in N-code, which is read into the engine, compiled, and preserved as byte-code instructions for fast execution. TCP traffic is applied against a reassembly table which preserves the state of each current TCP session. The state reassembly mechanism permits matching patterns or other events within the lifetime of a TCP stream, and keeps statistics pertaining to variance in the delivery of packets. These statistics are used to determine when the engine will stop watching a given connection – for example, connections are not considered "closed" until a time-out has exceeded two standard deviations of the average packet arrival rate after a FIN packet. Certain types of broken packets can be detected, and users can access byte counts of retransmitted packets – duplication of traffic – which might indicate network problems.

The filtering language binds a filter to an event or the reception of a packet. Once a packet has been applied against the filters, it is discarded. The filtering language provides programmatic access to fields within the packet, usually for the purpose of recording information from or about the fields. The main primitives for getting data out of a filter are the *alert* and *record* mechanisms. Alerts pass a free-form message to the alert management system, much like a call to syslog(). The record mechanism passes a constructed data structure to a backend recorder for further processing. Using a structured record allows the engine and the backend to pre-agree upon what type of information the recorder should expect to read, permitting for faster interpretation of the data as it arrives at the backend. Some of the backends provided with NFR are specialized to handle multiple forms of data, but if someone wanted to develop a simple backend that processed only IP addresses, they would be able to link in a simple library routine that provides the

backend a stream of pre-processed addresses. This approach saved us considerable time in development, and permitted early experiments with backends written in TCL [12]. We feel this is an important capability, since it opens the possibility of having an NFR performing gross-level filtering of traffic while passing specific records into a backend SQL database for more advanced processing.

Backends

Originally, we had anticipated developing a large number of specialized backends, each of which might maintain a different type of statistic. As the design evolved, we chose instead to produce a small number of multi-purpose backends, which accept and process a wider latitude of data types. Histogram[3] and list are the two primary backends in use at this time. Histogram maintains a columnar table of data, either totaling discrete values in the columns or incrementing them.

Unlike the traditional histogram in statistics, the NFR histogram recorder saves data in an arbitrary number dimensions, rather than in a single dimension. For example, a histogram that stores a list of IP addresses and strings might be used to represent the number of times clients on a network retrieved a particular URL. A different instance of histogram with the same record structure might store the number of times user-ids had logged into various systems within the network. Histograms can store multiple columns of data simultaneously, permitting a lot of information to be stored in a very compact space. The histogram recorder provides a very flexible means of generating alerts based on the appearance of previously-unseen values, or values exceeding a specified range. For example, a histogram can easily produce a "new/duplicate IP address detector" by throwing hardware ethernet addresses and IP addresses into a histogram, then generating an alert whenever a new entry appears.

The list backend maintains chronological records, as opposed to histogram's additive records. The list backend provides more typical logging functions by not collapsing its columns into totals. In the histogram example of client IP addresses and URLs, if the same data record were sent to list, it would maintain a log of individual accesses to the URLs by client, over time. List's data storage tends to be less space efficient than histogram, since it maintains data about each record it is sent, rather than a total. The additional detail list maintains is important for some applications, since it lets you know not only how many events happened, but exactly when and in what order.

Query Backends

One of the problems we faced early on was figuring out how to get data out of the NFR without interrupting the flow of data *in*. If the engine were blocked during a query, it might lose a large number of packets, since it's possible that a query against a large dataset might take several seconds to process. The eventual design decision was to have the

[3]Apparently we are using the term "histogram" somewhat incorrectly and may confuse statisticians. We should have called it "spreadsheet."

```
Time TCP Hash Client       Server           Command
Mon Sep  8 15:02:08 1997
     18 208.218.124.77  208.218.124.42  GET / HTTP/1.0
     18 208.218.124.77  208.218.124.42  If-Modified-Since: Wednesday, 06-Nov-96
                                            12:32:03 GMT; length=530
     18 208.218.124.77  208.218.124.42  Connection: Keep-Alive
     18 208.218.124.77  208.218.124.42  User-Agent: Mozilla/3.0Gold (X11; I;
                                            BSD/OS 3.0 i386)
     18 208.218.124.77  208.218.124.42  Host: cornfed
     18 208.218.124.77  208.218.124.42  Accept: image/gif, image/x-xbitmap,
                                            image/jpeg, image/pjpeg, */*
     18 208.218.124.77  208.218.124.42
     18 208.218.124.77  208.218.124.42  GET /apache_pb.gif HTTP/1.0
     18 208.218.124.77  208.218.124.42  If-Modified-Since: Wednesday, 03-Jul-96
                                            06:18:15 GMT; length=2326
     18 208.218.124.77  208.218.124.42  Referer: http://cornfed/
     18 208.218.124.77  208.218.124.42  Connection: Keep-Alive
     18 208.218.124.77  208.218.124.42  User-Agent: Mozilla/3.0Gold (X11; I;
                                            BSD/OS 3.0 i386)
     18 208.218.124.77  208.218.124.42  Host: cornfed
     18 208.218.124.77  208.218.124.42  Accept: image/gif, image/x-xbitmap,
                                            image/jpeg, image/pjpeg

     18 208.218.124.77  208.218.124.42
```

Figure 1: Historical data from a list recorder.

backends maintain their datasets entirely on disk, and to have a secondary backend – the *query backend* – which contains the logic for mining datasets produced by its matching backend. A backend may perform buffering of its own data, but since the query backend "knows" how the backend operates it can coordinate with the backend as necessary. Since one of the goals of NFR is to provide a reliable history of events, we feel that too much buffering is risky – we'd rather buy faster disks.

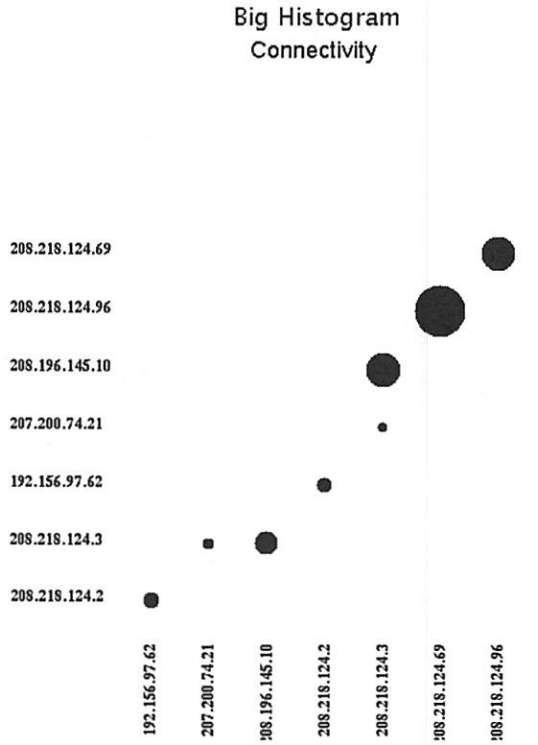

Figure 2: Connectivity graph with 10% cutoff in effect.

Each of the query backends provides its own CGI interface, parsing a URL into a query. Histogram and list both support options to compact out fields and eventually will support sorting of fields. Another feature we added later is threshold cutoffs – we discovered that some queries produced much, much more information than we could process sensibly. For example, an origin/source map between any two systems that have sent packets produces a scatter-plot that contains a large number of single-packet events (mostly DNS traffic). While the information is worth having, it is much easier to glance at when there is a cutoff in place. (See Figure 1)

The first version of our query backends represented information uniformly, which we found made it a bit more difficult to visualize. Since the data going into the backends is abstract, we needed to provide a mechanism for the end user to apply "eye candy" to the dataset to make it more comprehensible. User-specified mappings are matched against tokens in the output fields of histogram, changing color or replacing the strings. (See Figure 2)

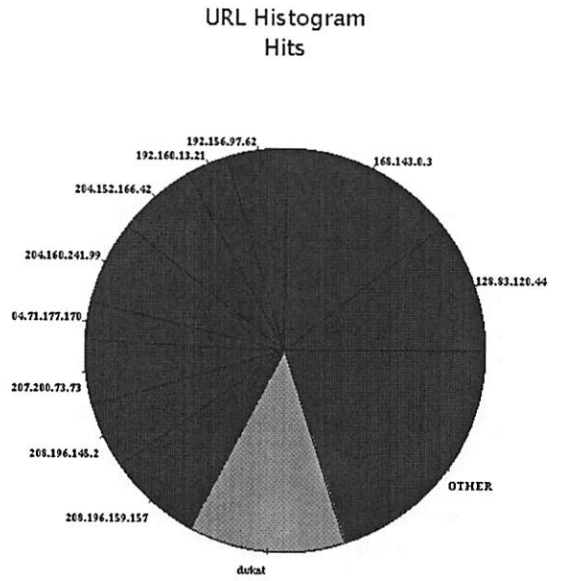

Figure 3: Pie chart with user-specified color and name mapping.

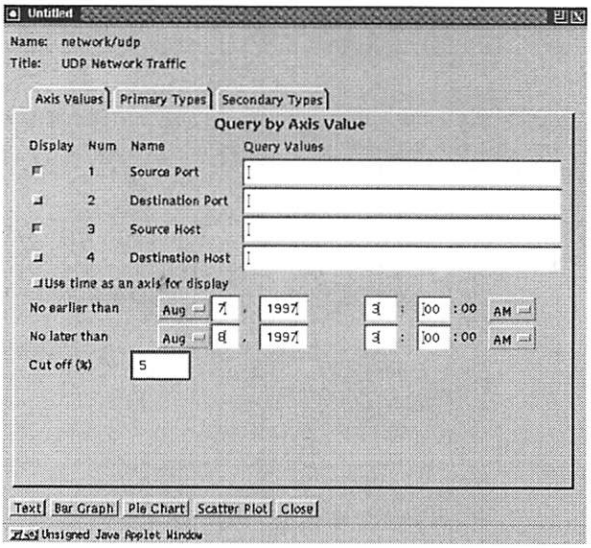

Figure 4: Query interface showing field collapsing.

We intend to continue to improve the query interface; that is an area of ongoing research. Currently, the query interface allows the user to specify field values and fields to represent, as well as how to represent them. Since each backend and query backend is a separate software system, there is some duplication of effort; we considered the problem of developing a generalized query language but concluded that the level of effort was too high. General query capability is best provided through an external database with SQL. Using an SQL database will greatly increase the cost, administrative cost, and setup cost of the system

in return for a marginally greater flexibility in query capability. We can imagine some cases where advanced query capability will justify the cost and effort, but we didn't want to try to integrate or develop a database engine as part of the project.

GUI

Each backend has a number of graphical user interface elements. The GUI elements for each backend control its individual configuration attributes, as well as its query interface. The query interface generates correctly configured CGI queries aimed at its matching backend. To simplify the user's view of the system, we've created a construct, which exists purely in the GUI, of "packages." Packages are a grouping of backends which may operate independently or may cooperate by sharing common code or variables. The latter case is particularly useful if you have a filter that does substantial work to analyze a protocol, and you want to record more than one type of information about it. For example, you may wish to analyze E-mail traffic and record sender/recipient to a list log and a matching count to a histogram, without having to duplicate the filter that performs the analysis.

While the Java language holds great promise, we found that Java is not yet ready for use in developing serious applications. Primarily, the difficulty is that the client browser implementations perform badly or unpredictably. While the core language itself appears solid, the browser supplied functions, which include all of the GUI objects and network communications, often behave erratically. Eventually, we developed a stable base of classes that seem to work on popular platforms, but we did it by testing the target systems to determine what parts of the published interface were actually reliable. The process was not as fun or pleasant as the Java hype would lead one to expect.

For a program like NFR, it was vital that we be able to have a secured remote management capability. Using Java and Web technology allowed us to layer the NFR interface underneath whatever security the browser and web server can provide. This freed us from worrying about export control regulations that apply to software incorporating cryptography. It also freed us from portability problems, in a conceptual sense, though the browser-version-specific bugs we coded around were another form of portability problem. "Write once, run anywhere" does not promise that your code will actually *work* once it is running. In spite of our difficulties with Java, it was still easier to implement our U/I than it would have been had we been using Microsoft Windows and X Toolkit to do our own secure remote user interface. Java has a much simpler GUI model than either X or Windows, but, when it works, it is suitable for real applications. We hope that future versions of Java, such as Java 1.1, will overcome some of the difficulties we faced.

Alert Queue Manager

NFR has a generalized mechanism for processing alerts, which is used to manage alerts that are produced from N-code, alerts from backends, and internal error-alerts generated by the engine. In order to make NFR a useful network management tool, we developed alertd, which prioritizes and routes alerts. Alerts can include free-form or formatted strings, which are matched against a number of delivery facilities. The facilities represent delivery mechanisms such as printer, E-mail, and FAX. Alert events can further be marked as alerts requiring acknowledgment, which maintains them in a queue that a network manager can clear once the situation causing the alert has been resolved. Records of who acknowledged an alert, as well as comments, are preserved with the alert.

Space Manager

Since NFR is intended to provide historical records, we needed to provide a capability for rotating logs and archiving them or deleting them. Spaceman, the space manager, runs periodically to check the amount of disk space available for data storage, and to "expire" old datasets. Each instance of a backend can be specifically tuned for storage lifetime and archiving. Backends that are preserving especially important data might have an expiration time that is much higher, and an archiving process that copies the data to CD-R or tape instead of merely deleting it.

N-code Filtering

The N programming language is a derivation of an interpreted language designed years ago for use in a computer game.[4] The interpreter operates on a byte-code instruction set that implements a simple stack machine. One advantage of this approach is that NFR filters occupy very little memory, yet are quite fast to evaluate. N is a complete programming language including flow control, procedures, variables with scoping rules, and list data types. Unlike many programming languages, however, N has primary data types such as "IP address." Since NFRs may be used on large networks, we chose to implement counter data types as 64-bit integers, to reduce the chance of overflow.

Packet values are accessible in N-code using a syntax of *thing.thing*, in which a higher level attribute references a lower level element. For example: *ip.src*, *ip.dst*, *syslog.message*, and *tcp.hdr* are all valid N-code packet fields. The packet ripping routines perform lazy evaluation with cached responses, so the runtime cost of packet ripping is kept low. At present, the NFR engine's packet rippers can handle IP, TCP, and UDP packets and ethernet frames. We are considering adding routines to decode other protocols or even application protocols, such as *syslog*. Mixing

[4] UberMUD, a meta-programmed multi-user dungeon game by Marcus J. Ranum

application specific protocols into packet value access enables some unusually powerful productions such as:

```
if (ip.src == $myfirewall &&
    index(syslog.message,"BADSU")
        >= 0 ) { ...
```

In cases where the built-in packet fields are insufficient, other fields can be located by N-code functions. It is possible, though slightly slower, to write N-code subroutines to return byte offsets within packets. This can extend as far as analyzing high-level protocols. For example, we have implemented an N-code filter that extracts protocol fields such as sender and recipient from an SMTP dialog. More N-code can then act on that information:

```
if ( $sender == $known_spammers )
    { ...
if ( $n_recipients > 10000 )
    { ... / possible spam
```

Events

Triggers within N-code occur upon receipt or detection of an event that the code is attached to. Events can be triggered with limitations on source, destination, ports, client or server side (if known), or patterns within the TCP stream. The syntax looks like:

```
filter mailtrack tcp ( client,
    dport: 25 ) {
```

The filter above is a simple TCP stream trigger that will monitor the client side of SMTP connections. The "client" and "server" notion is based on the reassembly engine's recollection of which system initiated the connection that is being observed.

Keywords that can be placed within an event are:

- client – from the caller
- server – from the called
- start: "string" – begin matching
- stop: "string" – end matching
- opensession – on start of connection
- closesession – on end of connection
- port – IP port number (source or dest)
- sport – source port
- dport – destination port
- host – source or destination address
- net – source or destination network
- dst – destination address
- src – source address

A typical use is to configure an event to call N code for as small a subset of received data as is practical, then implement any further filtering in N code. To detect spam, for example, you might select TCP traffic for port 25/SMTP. The N code would then:

- Observe the transaction and determine the sender and recipient
- If the sender is the spammer, record the sender, recipient, and originating host to a histogram recorder
- If the sender is the spammer, record the message header and the first 10 lines of the message text to a log recorder

```
filter server tcp ( client,
        port: 80,
        start: "GET ",
        stop: " " ) {
    record  ip.src, ip.dst,
        tcp.sport, tcp.dport,
        tcp.bytes to urlRecorder;
}
```

Figure 4: N-code implementing a simple HTTP URL detector.

The example HTTP URL detector sets a trigger filter on TCP port 80, for patterns that look like HTTP GET commands, and logs information about the activity to a recorder. In the example, tcp.bytes represents the matched TCP data between the start and stop triggers. More complex filters can attach symbolic values as local variables for each TCP stream – permitting a filter to do complex activity like counting the To: recipients in an SMTP stream, or recording them in an array for later logging. The N-code for more advanced filters, with comments, can run to several hundred lines.

Performance

Performance is an open question with an NFR. We haven't had enough access (yet) to truly large networks that can properly stress the system. For some reason, sites with large, interesting, backbones are not enthusiastic about having someone else put a meta-programmable traffic analysis engine on their network. As of this writing we have not had a chance to test the NFR software on more than 10Mb/s ethernet. We believe that performance should be adequate up to at least 60-70Mb/s, based on experiments other sites have performed with TCPdump on 100-base-T hubs, using comparable hardware.

Probably the biggest performance problem that we worry about is the NFR's programmability. Since it is highly customizable, it's hard to predict how many filters the average user will install, and how much data they will be recording. For a network that runs a steady 80Mb/s load, with a filter recording all the packets to disk, the NFR would not be able to keep up, unless the I/O subsystem were capable of sustained disk writes, through the filesystem, at comparable speeds. We've found that, in general, filters compile down to be quite tiny and require few "instructions" to operate, but it's conceivable that a user might develop an extremely N-code-intensive filter which bogged a system down unacceptably. We fully anticipate that this will happen, and plan to performance tune the system as we get more feedback from installed sites.

Lessons Learned

The first hurdle in designing the system was figuring out the right place to put each piece of functionality. TCP reassembly, for example, was originally planned for implementation in a backend, but we realized that some of the filters we wanted to install depended on reassembled streams. TCP reassembly moved back into the engine. Some of the statistical capabilities of the backends were originally slated for residence in the engine, but we realized that queries against the engine itself might cause it to spend too long processing the query and lose packets. As a result, the backends show duplication of functionality, but are standalone programs. The decision to separate querying completely from the backend into a separate program was risky but has proven to work fairly well at the cost of duplication of functionality between the query backend and the recorder backend.

The most hard-won lesson of the project, so far, is that Java isn't really ready for writing major user interfaces. It's great for nifty animated objects, but until JDK1.1 – and widely supported 1.1 virtual machines – are available, Java will be a problem. We also learned that designing a general-purpose language for traffic analysis was harder than we originally expected it to be. We began with a packet-oriented event model, which did not survive well in the face of TCP streams and matching. NFR seems to be evolving toward a model in which filters are callbacks triggered by pre-specified conditions within the engine. Now that we've tried it the other way, it makes a lot of sense!

Future Work

In future releases of NFR we intend to expand the engine's ability to rip apart packets and application traffic. As we discover more types of things we want to record, it is likely we'll extend the language. Doubtless we will have cause to seek and find ways to improve the NFR's performance; there is no such thing as "fast enough." We believe that NFR will make a useful "bottom half" of an intrusion detection system, and are considering a number of options for analysis engines that can be plugged into NFR as a backend. Another area to explore is the development of active agents which will "poke" parts of the network so that NFR can record the responses.

Conclusions

We set out to build an extremely flexible general-purpose security and network management tool, and appear to have met our goal. There is still a great deal of enhancement we hope to make to the system, but the basic functionality already exceeds what we had originally expected. In a few areas, we were surprised that things we thought would be very useful turned out to be less so – a good example is the scatter plot diagrams prior to the addition of cutoff thresholds.

Initially, we were concerned about the possibility for abusing our tool. It would, for example, make a terrific password grabber. But we feel it is too "heavyweight" for such uses – most hackers who are grabbing passwords are already using very simple single-purpose software that does the job quite nicely. NFR presents the possibility for abuse by "big brother" or well-funded snoops monitoring networks on a grand scale. Keeping NFR out of the public's hand will not prevent such activities, either, since specialized tools for particular traffic analysis are not difficult to develop. We believe that, in general, NFR will be a net benefit to the community, by providing a tool that will stimulate research in network analysis and intrusion detection.

We've been very, very pleased with the power of the system; it appears to be able to do all of the kinds of things for which we designed it. Simple statistics and traffic gathering is very straightforward, and very sophisticated programs are possible as well. We haven't really had time to explore all the possibilities of the system, and we see the depth of its capabilities as a great source for future research and discovery.

Availability

The complete NFR source code, including documentation, Java class source, decision engine, space manager, etc., is available for download from www.nfr.net. for non-commercial research use. The code is designed to operate on a wide variety of UNIX platforms and is being ported to Windows NT for commercial release.

Acknowledgements

Marcus Ranum wrote the first version of the engine interpreter and early syntax for the N language, as well as developing the overall early design of NFR. Mike Stolarchuk implemented the TCP reassembly code in the engine, and refined the engine and N language to its current state, patiently taking feedback from the rest of us. Mark Sienkiewicz implemented the backend design, including histogram, list, and the layout of many of the Java user interface elements. Kent Landfield implemented spaceman, and a number of the user interface elements. Andrew Lambeth designed and implemented alertd, and the list viewer applets. Eric Wall implemented further user interface elements, stealth mode kernel hacks, and did system configuration.

Network Flight Recorder would like to thank our investors for their kind support of our efforts.

References

[1] Robert Braden and Annette DeSchon, *NSFnet Statistics Collection System – NNStat*, USC Information Sciences Institute, December, 1992.

[2] *ARPwatch*, Lawrence Berkeley National Labs Network Research Group, http://ftp.ee.lbl.gov .

[3] *TCPwatch*, Lawrence Berkeley National Labs Network Research Group, http://ftp.ee.lbl.gov .

[4] *Homebrew Network Monitoring: a Prelude to Network Management*, Mike Schultze, George

Benko, and Craig Farrell. Curtin University of Technology, 1993

[5] Clog, Brian Mitchell.

[6] *Netwatch and Netwatch*, Texas A&M University, January 1994.

[7] Carter Bullard, Chas DiFatta, *Argus 1.5 announcement*, Software Engineering Institute, Carnegie Mellon University, ftp://lancaster.andrew.cmu.edu/pub/argus-1.5 .

[8] *Tcp_Wrappers and Logdaemon*, Wietse Venema.

[9] *Klaxon*, Doug Hughes.

[10] *Tocsin*, Doug Hughes.

[11] *libpcap*, Lawrence Berkeley National Labs Network Research Group, http://ftp.ee.lbl.gov .

[12] Ousterhout, J. K., "TCL: An Embeddable Command Language," *Proceedings of the 1990 Winter USENIX Conference*, pp 133-146, 1990.

[13] Anderson and Patterson, "Extensible, Scalable Monitoring for Clusters of Computers," *Proceedings of the 1997 USENIX LISA Conference*, 1997.

Appendix 1: A Simple Filter

```
# This filter serves two purposes:  to record client requests
# made to your web servers, and to serve as example in the LISA paper.
#       Mark Sienkiewicz / NFR
#       Copyright 1997, Network Flight Recorder, Inc.

# schema would be automatically generated by a "wizard"
watchservers_schema = [ 1, 1, 1, 6, 6, 2 ];

# list of my web servers
my_web_servers = [ 208.218.124.77 , 208.218.124.42 ] ;

# gather data the client sends to a web server.  If I didn't know that
# all my web servers are on port 80 I would make this more elaborate.
filter watch tcp ( client, dport: 80 )
{
        if (ip.dest != my_web_servers)
                return;
        declare $blob inside tcp.connSym;
        $blob = strcat ( $blob, tcp.bytes );
        while (1 == 1)
                {
                $x = index( $blob, "\n" );
                if ($x < 0)             # break loop if no complete line yet
                        break;
                $t=substr($blob,$x-1,1);        # look for cr at end of line
                if ($t == '\r')
                        $t=substr($blob,0,$x-1);        # tear off line
                else
                        $t=substr($blob,0,$x);

                # save the time, the connection hash, the client,
                # the server, and the command to a list
                record system.time, tcp.connHash, ip.src, ip.dest, $t to
                        watchservers_list;

                # keep the remainder of the blob for the next pass
                $blob = substr($blob, $x + 1);
                }

        # keep us from getting flooded if there is no newline in the data
        if (strlen($blob) > 4096)
                $blob = "";
}

watchservers_list = recorder ("bin/list packages/web/watchservers.cfg",
        "watchservers_schema" );
```

Extensible, Scalable Monitoring for Clusters of Computers

Eric Anderson and Dave Patterson – U. C. Berkeley

ABSTRACT

We describe the CARD (Cluster Administration using Relational Databases) system[1] for monitoring large clusters of cooperating computers. CARD scales both in capacity and in visualization to at least 150 machines, and can in principle scale far beyond that. The architecture is easily extensible to monitor new cluster software and hardware. CARD detects and automatically recovers from common faults. CARD uses a Java applet as its primary interface allowing users anywhere in the world to monitor the cluster through their browser.

Introduction

Monitoring a large cluster of cooperating computers requires extensibility, fault tolerance, and scalability. We handle the evolution of software and hardware in our cluster by using relational tables to make CARD extensible. We detect and recover from node and network failures by using timestamps to resynchronize out system. We improve data scalability by using a hierarchy of databases and a hybrid push/pull protocol for efficiently delivering data from sources to sinks. Finally, we improve visualization scalability by statistical aggregation and using color to reduce information loss.

We have designed and implemented a system for flexible, online gathering and visualization of statistics and textual information from hundreds of data sources. CARD gathers node statistics such as CPU and disk usage, and node information such as executing processes. We will describe the system we developed to monitor our cluster, explain how we solved the four problems described above, and show how we synthesized research from other fields and applied them to our problem.

To make CARD flexible and extensible, we have chosen to gather data and store it in a relational database [Codd76]. New subsystems can access the data through SQL [Cham76] without requiring modification of old programs. We use SQL to both execute ad-hoc queries over the database, and to extract data for visualization in our Java [Gosl95] applet. Relational tables make CARD naturally extensible because new data can go into a new table without affecting old tables. In addition, new columns can be added to tables without breaking older programs. The column names also help users understand the structure of the data when browsing. We have used additional tables of descriptions to further assist in browsing.

We use timestamps to detect and recover from failures in CARD and the cluster. Failures are detected when periodic data updates stop. Changing data is synchronized using timestamp consistency control. Stale data is expired when the timestamps are too old.

We have achieved data scalability, which is the ability to handle more data as machines are added, by building a hierarchy of databases. The hierarchy allows us to batch updates to the database, specialize nodes to interesting subsets of the data, and reduce the frequency of updates to the higher level nodes. This increases the scalability of the database, but the last two approaches reduce either the scope or the freshness of the data. Users of the data may then need to contact multiple databases, to gain data coverage or to get the most up to date information.

We have improved the network efficiency, and hence the data scalability by creating a hybrid push-pull protocol for moving data from sources to sinks. Our protocol sends an initial SQL request and a repeat rate. The query is executed repeatedly, and the results are forwarded to the requestor. The hybrid protocol achieves the best of both a request-response (pull) protocol and an update (push) protocol.

We increase visualization scalability, which is the ability to gracefully increase the amount of data displayed without having to increase the screen space, through statistical aggregation of data, and the resulting information loss is reduced by using different shades of the same color to display dispersion. These two techniques have allowed us to meaningfully display multiple statistics from hundreds of machines.

The remainder of the paper is structured as follows. The next section describes our four solutions, the Implementation section describes our experience with our implementation, and the next section describes the related work. The last section summarizes our conclusions from building the system.

[1]CARD is available from http://now.cs.berkeley.edu/Sysadmin/esm/intro.html .

Four Problems and Our Solutions

We will now describe our solutions to four problems of monitoring large clusters. First, we will explain how we handle the evolution of software and hardware in a cluster. Second, we will explain how we deal with failures in the cluster and our software. Third, we will explain how we increase data scalability. Fourth, we will explain how we display the statistics and information from hundreds of machines.

Handling Rapid Evolution using Relational Databases

Cluster software is evolving at a rapid pace, so a monitoring system needs to be extensible to keep up with the changes. This means that new data will be placed in the system, and usage of the data will change. Hence a system with only one way of storing or querying data will have trouble adapting to new uses.

We believe that flexibility and extensibility can be achieved by using a relational database to store all of the data. The database increases flexibility by decoupling the data users from the data providers, which means that arbitrary processes can easily put information into the database, and arbitrary consumers can extract the data from the system. The database also improves flexibility by supporting SQL queries over the data. SQL queries can combine arbitrary tables and columns in many ways, and the database will automatically use indices to execute the queries efficiently. As the use of the database changes, new indices can be added to maintain the efficiency of the queries. The database increases extensibility because new tables can be easily added, and new columns can be added to old tables without breaking old applications. Queries only address columns in tables by name, and hence the new columns do not affect the old queries.

Using a database is a significant departure from previous systems [Apis96, Dolphin96, Fink97, Hans93, Hard92, Scha93, Schö93, Seda95, Ship91, Simo91, Walt95], which all use a custom module for data storage and few provide any external access to the data. While building an integrated module can increase efficiency for a single consumer of the data, some of that improvement is lost with multiple consumers. Furthermore, the flexibility of the system is reduced because adding new data producers and consumers is more difficult.

Recovering from Failures using Timestamps

The second problem we address is detecting and recovering from failures. We use timestamps to detect when parts of the system are not working, identify when data has changed, and determine when data has become old and should be rechecked or removed.

Timestamps help detect failures when data sources are generating periodic updates. If the timestamp associated with the data is not changing, then the check has failed, which indicates that the remote node is either slow or broken. This solution works even if the updates are propagating through multiple databases in the hierarchy because the timestamps are associated with the data and don't change as the data moves.

We also use timestamps for consistency control [Chan85]. Timestamps allow quick comparisons of data to determine if it has been updated. We have a timestamp associated with both the data, and the time for when the data was placed in the database. Remote processes maintain a last timestamp (t_0). To synchronize with the database, they get a new timestamp from the database (t_1), get all the data that was added since t_0, and set t_0 to t_1. By repeating this process, the remote process can be kept weakly synchronized with the database. Moreover, if the machines' time is synchronized [Mill95], then the remote process also knows the freshness of their data. Timestamp consistency control is very simple to implement in comparison to other consistency protocols [Gray75], and if the database is accessible, then the data is always available regardless of other failures in the system, whereas other protocols may deny access to ensure stricter consistency.

Finally, we use timestamps to eliminate stale data. Stale data can occur because of failures or removals. The timestamps allow the data to be automatically removed after a table specific period, which

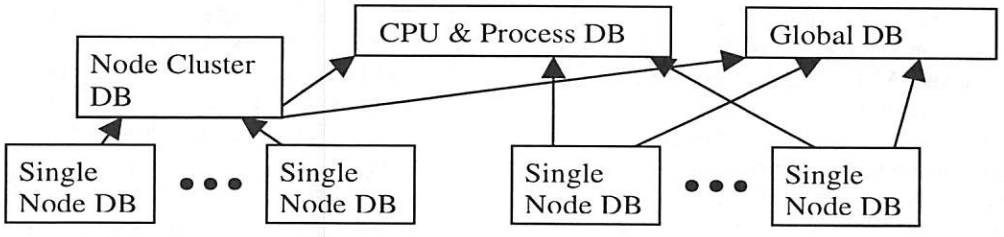

Figure 1: A hierarchy of databases. At the lowest level are single node databases. These hold information gathered from a single node. The top level shows a few forms of specialization. The node cluster database gathers information about all the single nodes in its cluster. The CPU and process database stores a subset of the data at the full frequency and takes advantage of the batching possible because of the node cluster database. The global database stores all the information about the cluster at a reduced frequency.

means that the system will automatically recover to a stable state. Multiple timers allow slowly changing data like physical memory to be updated infrequently yet not be declared stale.

Data Scalability using Hierarchy

Systems that can take advantage of multiple machines are usually more scalable. We have designed a hierarchy (Figure 1) of databases in order to make our system more scalable. The hierarchy provides many benefits.

A hierarchy allows updates to a database to be batched. This reduces the network overhead by reducing the number of packets that need to be transmitted. This allows more operations because the operations are not serialized and hence the latency of the network is unimportant. Finally, this allows the database to perform more efficient updates.

A hierarchy also allows specialization of nodes. While a single database may not be able to handle the full update rate for all of the information that is being collected, a single database may be able to handle a useful subset of the data at the full rate. For example, statistics about CPU usage and processes could be stored in a single database, allowing it to handle more nodes.

Furthermore, a hierarchy allows reduction in the data rate. For example, an upper level database could keep all of the information gathered at an interval of a few minutes. As the amount of data gathered grows, the interval can be reduced. This will allow a single database to always keep a complete, but more slowly changing, copy of the database.

Finally, a hierarchy over multiple machines allows for fault tolerance. Multiple databases can be storing the same information, and hence if one of the databases crashes, other nodes will still have access to the data.

Data Transfer Efficiency using a Hybrid Push/Pull Protocol

Given a hierarchy of databases, and a number of additional processes which are also accessing the data, the system needs an efficient method for transferring data from sources to sinks. Most systems use polling; a few also have occasional updates. Both approaches have disadvantages, so we have developed a hybrid push-pull protocol that minimizes wasted data delivery, maximizes freshness, and reduces network traffic.

The canonical pull protocol is RPC [Sun86]; SNMP is a mostly pull protocol. A pull-based system requires the sink to request every piece of data it wants from the source. If the sink wants regular updates, it polls the source. Since the source knows it wants regular updates, all of the request packets are wasted network bandwidth. Furthermore, if the data is changing slowly or irregularly, some of the polls will return duplicates or no data. However, polling has the advantage that since the data was requested, the sink almost always wants the data when it arrives. Between polls, the data goes steadily out of date, leading to a tradeoff between the guaranteed freshness of the data and the wasted traffic.

PointCast [Poin97] and Marimba Castanet [Mari97] use a push protocol. A push protocol delivers data all the time and allows sinks to ignore data if they don't want it. Multicast [Deer90] uses a pruned push model, and broadcast disks [Acha97] use a push model with broadcasts to all receivers. A push system is ideal when the sink's needs match the source's schedule since the data is current, and the network traffic is reduced because the sink is not generating requests. However, as the sink's needs diverge from the source's schedule, a push model begins delivering data that is wasted because the sink will just ignore it. Also, sinks have to wait until the source decides to retransmit in order to get data. The push model trades off between wasted transmissions and communication usage.

We use a hybrid model. Sinks send an SQL request to the forwarder along with a count, and an interval. The forwarder will execute the query until it has sent count updates to the requester. The forwarder will suppress updates that occur more frequently than interval. By setting the count to one, a sink can get the effect of the pull model. By setting the count to infinity, a sink can get the effect of the push model. Intermediate values allow requesters to avoid being overrun with updates by requiring them to occasionally refresh the query. The interval allows the requester to reduce the rate of updates. When the updates are no longer needed, the sink can cancel the request. In both cases, the sink can use the full power of SQL to precisely retrieve the desired data. Using SQL reduces one of the main disadvantages of the push model, which is that the data delivered is not the desired data. For example, in multicast, the only selection that can be made is the multicast address, no sub-selection of packets from particular sources can be made.

Visualization Scalability Using Aggregation

We have found that we need to use aggregation to scale the visualization to our whole cluster. We tried having a stripchart for every statistic we wanted, but ran out of screen space trying to display all of our machines. We therefore aggregate statistics in two ways: First, we combine across the same statistics for different nodes. For example, we calculate the average and the standard deviation across the CPU usage for a set of nodes. We then display the average as the height in the stripchart, and the standard deviation as the shade. Second, we aggregate across different statistics. For example, we combine together CPU, disk and network utilization to get a single statistic we call machine utilization. Using shade takes advantage of the eye's ability to perceive many different shades of a color [Murc84, HSV]. We also use color to help draw distinctions and identify important information. For

example, we use different colors for the I/O and user CPU usage, and we identify groups of down machines in red. Figure 2 shows a use of the first form of aggregation, and the uses of color as it is a snapshot of our system in use.

Implementation & Experience

We chose MiniSQL [Hugh97] for our SQL database. MiniSQL is freely available to universities, and has a very slight license for companies. The MiniSQL database is sufficient for our needs because we don't require complex SQL queries. In addition, because MiniSQL comes with source, we were able to extend it when necessary. For example, we added micro-second timestamps to the database so that we could extract data changing on a short time-scale. We also removed the need to wait for a response from the server after an SQL command, which allows groups of updates to be sent as a batch. Our use of MiniSQL also allows other sites to try our system without an expensive initial investment in a database.

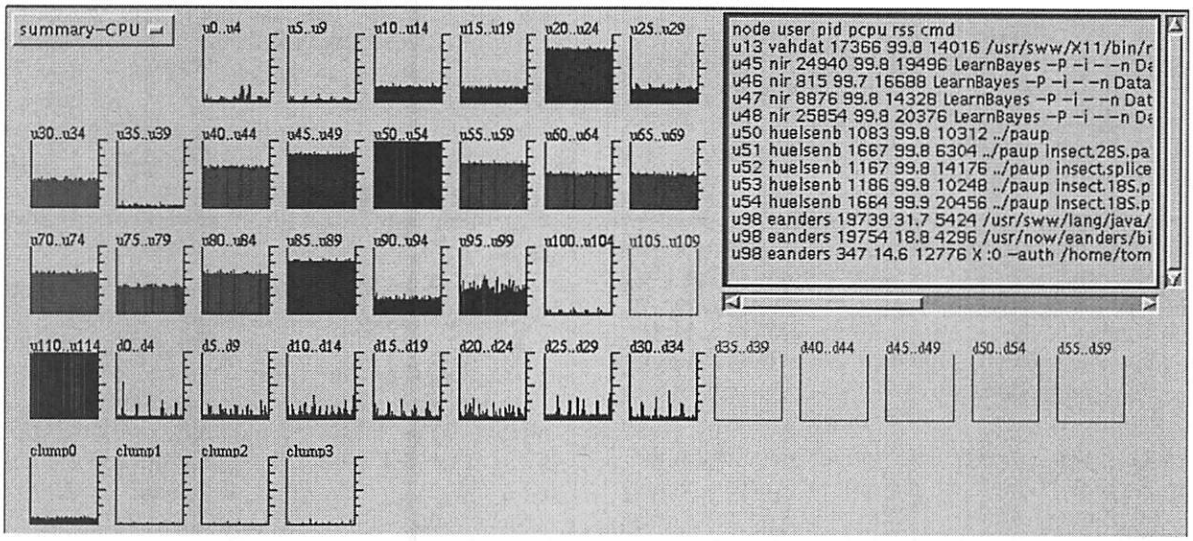

Figure 2: Snapshot of the Java interface monitoring our entire cluster which consists of 115 Ultra 1's, 45 Sparc 10's or 20's, and 4 Enterprise 5000 SMP's. Aggregration has been done with averages (height) and standard deviation (shade) across groups of 5 machines except for SMP's. The darker charts are more balanced across the group (u50..u54) all are at 100%), and the lighter charts are less balanced (u40..u44 have three nodes at 100% since the average is 60% and the usage is not balanced). All charts show the system CPU time in blue over the green user CPU time; u13 has a runaway Netscape process on it. Nodes u105-u109 and dawn35-dawn59 are down and shown in red. Processes running on selected nodes are shown in the text box in the upper right hand corner. A color version of this chart is available as http://now.cs.berkeley.edu/Sysadmin/esm/javadc.gif.

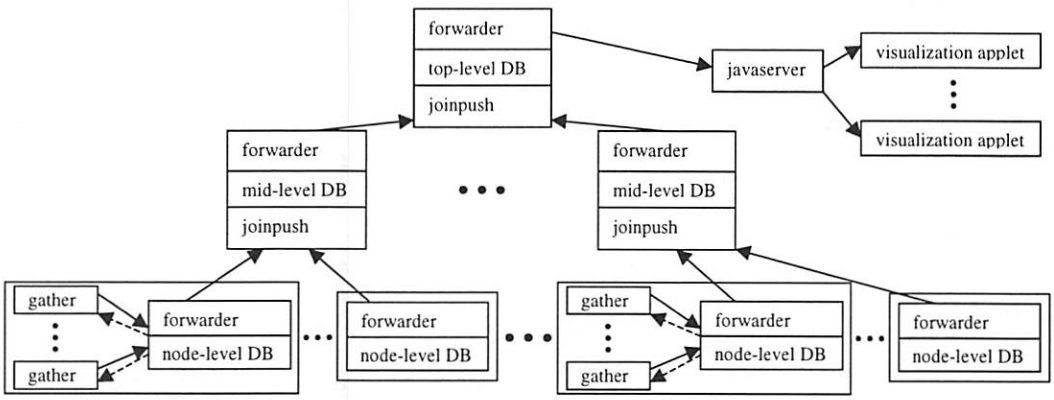

Figure 3: Architecture of our system. The gather processes are replicated for each forwarder/node-level database (DB) group. The top level databases can also be replicated for fault tolerance or scalability. The javaserver acts as a network proxy for the visualization applets because they can not make arbitrary network connections. The forwarder and joinpush processes are associated with a database and serve as the plumbing that moves data through the system.

Our experience using an SQL database has been very positive. We initially stored all of our configuration information in separate, per-program files. Given our experience with the reliability, and flexibility of the database, we have now moved the configuration information into the database.

Figure 3 shows the data flow among the major components of our system. The forwarder process, associated with each database, accepts SQL requests from sinks, executes them periodically and forwards the results to the sinks. The joinpush process merges the updates pushed from the forwarder processes into an upper-level database. The javaserver process acts as a network proxy for the Java visualization applet because applets cannot make arbitrary network connections. The visualization applet accepts updates from the javaserver, and displays them in stripcharts or a text window for the user. The applet also provides a simple way to select a pane of information to view.

Resource usage of MiniSQL has not been a problem. The database uses 1-2% of the CPU on an Ultra 1/170 to update 20-30 statistics a second. Our upper level databases seem to peak somewhere between 1500 and 2000 updates/second. There are many optimizations we could make to reduce or rebalance the load if necessary; we have already taken advantage of the indexing support in MiniSQL.

Initially we were not concerned with getting our system running, however we discovered after using it for a little while that making sure all of the pieces were functioning correctly, especially when we were changing parts, was tricky. We therefore created a process which watches to see if the various components on a node are reachable, and if they are not attempts to restart the node. We discovered that making sure that leftover processes didn't accumulate required careful design. We use two methods to ensure that old processes terminate quickly. First, each process creates a pid file and locks the file. The reset-node operation attempts to lock each file and if the lock fails, the process is sent a signal. Using a lock guarantees that we won't accidentally try to kill off a second process which just happens to have the same process id. Second, we write an epoch file each time the node is reset. Processes can check the epoch file, and if it doesn't match then they know they should exit. We added the second approach because we occasionally saw processes not exit despite having been sent a signal that should cause them to exit.

The forwarder, and joinpush processes are both implemented in C taking advantage of Solaris threads in order to achieve better efficiency. We initially tried implementing those processes in Perl, but the code was too inefficient to support 150 nodes, and using threads reduced concerns about blocking while reconnecting. The javaserver is implemented in Perl [Wall96] to simplify implementation and to support run-time extension. The applets are implemented in Java so that users can access our system without having to download or compile anything other than a Java enabled browser.

Data is added to the system by the gather process, which is also implemented in Perl. We originally examined a threaded implementation of the gather process, but we were unable to get a sufficiently stable multi-threaded implementation. We therefore use a multi-process implementation. We have a directory of files which all get spawned by the first process, which then waits around and flushes old data from the database. We currently have threads that extract CPU, I/O, network, process and machine statistics. We also have a specialized thread for extracting information about our high-speed Myrinet [Myri96] network and our lightweight Active Messages implementation [Chun97].

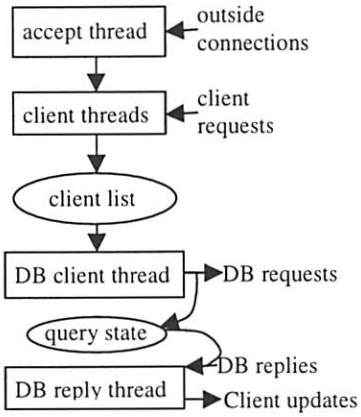

Figure 4: Architecture of the forwarder. The left column shows either threads or important data structures in the forwarder. The right column shows the interactions with other processes.

Figure 4 shows the architecture of the forwarder. The accept thread gets outside connections from clients and immediately forks a client thread to deal with each client. The client threads take requests for updates from the clients, and put those requests in the structure associated with each client. The use of threads allows us to easily handle slow clients without concerns of blocking. The database client thread walks the list of clients, and issues the requests which are pending to the database. When the response comes back from the database, it is matched with the information stored at request time, and the reply thread sends the updates to the appropriate client.

Figure 5 shows the architecture of the joinpush process. The list of forwarders and the data to request is configured externally to joinpush to simplify the implementation. The reconnect thread forks separate threads to connect to each of the forwarders and issue a request. When a connection is made, the connection is added to the connections list, and the update thread waits around for updates from any of the forwarders. It generates an appropriate database update. The reply

thread will generate an insert request if the reply indicates that the data was not yet in the table.

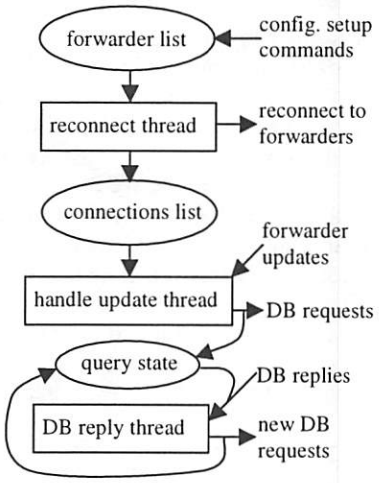

Figure 5: Architecture of joinpush. The left column shows either threads or important data structures in joinpush. The right column shows the interactions with other processes. The configuration commands are handled analogously to how clients are handled in the forwarder.

We have found the timestamps associated with the data to be extremely useful. For example, an early version of CARD failed to send the timestamp from the javaserver to the visualization applet. When the javaserver generated repeated updates of old data, the client was fooled into thinking the system was performing correctly. Now that we forward timestamps, we would detect this error.

Because MiniSQL doesn't support triggers [Eswa76], our actual implementation of the forwarder simply executes the query count times every interval seconds. This implementation is a reasonable approximation of the more optimal hybrid protocol. The main disadvantage is that the forwarder will sometimes execute the query and there will have been no updates. Since the forwarder is on the same node as the database, and the query restricts the results to new data, this is a fairly minor problem. A more optimal implementation would merge the forwarder and the database into the same process.

The fact that we display information through a Java applet raises a few privacy concerns. In particular, outside users can see all of the statistics of the cluster. Given that we are an academic institution, we have not been very concerned about maintaining secrecy of our usage information. However, all of the standard techniques for improving privacy could be added to our system. For example, access could be limited by IP address, or secure, authenticated connections could be established via the secure socket layer [Frei96]. To ensure privacy, it would be necessary to apply this protection to the javaserver, the forwarder, and the MiniSQL server. To prevent bogus information from being added into the database, it might also be necessary to protect the joinpush process.

Related Work

The most closely related work is TkIned [Schö93, Schö97]. TkIned is a centralized system for managing networks. It has an extensive collection of methods for gathering data. Because it is distributed with complete source code, it can be extended by modifying the program. Since the data is not accessible outside of the TkIned program, new modules either have to be added to TkIned, or have to repeat the data gathering. TkIned provides simple support for visualization and does not aggregate data before displaying it. TkIned's centralized pull model limits its scalability.

Pulsar [Fink97] uses short scripts (pulse monitors) which measure a statistic, and send an update to a central display server if the value is out of some hard-coded bounds. Pulse monitors run infrequently out of a cron-like tool. Pulsar can be extended by writing additional pulse monitors, and adding them to a configuration file. Pulsar's centralized design is not fault tolerant, and only simple support for external access to updates. Pulsar does not support monitoring of rapidly changing statistics.

SunNet Manager [SNM] is a commercially supported, SNMP-based network monitoring program. Other companies can extend it by writing drop-in modules to manage their equipment. Using SNMP version 2 [Case96], or Sun proprietary protocols, it has some support for multiple monitoring stations to communicate with each other. As with other monolithic systems, it has poor scalability and weak extensibility.

The DEVise [Livn96, Livn97] system is a generic trace file visualization tool. DEVise supports converting a sequence of records (rows in a table) into a sequence of graphical object, displaying the graphical objects, and performing graphical queries on the objects. DEVise uses SQL queries to implement the graphical queries, and supports visualizing trace files larger than the physical memory on a machine. Unfortunately, it does not support online updates to visualized data, and so does not directly match our needs, but we are using a similar idea of translating database updates into graphical objects.

Conclusion

Decoupling data visualization and data gathering through the relational database has greatly improved the flexibility and structure of our system. It led to our success in using relational tables for flexible data storage and access. It also led to the idea of using a hierarchy and a hybrid protocol for efficient data transfer. Timestamps have been very useful in detecting internal system failures and automatically

recovering from them. Since the machines are used on a research project [Ande95] exploring how to use hundreds of machines in cooperation to solve complex problems, aggregation in visualization was required by the scale and class of the system we wanted to monitor. We expect in the future to monitor more of the software and hardware in our cluster, including more research systems. CARD is available for external use from http://now.cs.berkeley.edu/Sysadmin/esm/intro.html .

Acknowledgements

The authors would like to thank Remzi Arpaci-Dusseau, Armando Fox, Steve Gribble, Kim Keeton, Jeanna Matthews, Amin Vahdat, and Chad Yoshikawa for their help clarifying and condensing the important contributions of the paper.

Eric Anderson is supported by a Department of Defense, Office of Naval Research Fellowship. This work was also supported in part by the Defense Advanced Research Projects Agency (N00600-93-C-2481, F30602-95-C0014), the National Science Foundation (CDA 9401156), Sun Microsystems, California MICRO, Intel, Hewlett Packard, Microsoft and Mitsubishi.

Author Information

Eric A. Anderson (University of California at Berkeley) is a Ph.D. candidate in computer science working on System Administration. His thesis concerns monitoring and diagnosing problems in a cluster of computers. Eric's research interests include combining operating systems, distributed systems, and networks to build large systems capable of solving complex problems. Reach him electronically at <eanders@u98.cs.berkeley.edu> .

David A. Patterson (University of California at Berkeley) has taught computer architecture since joining the faculty in 1977, and is holder of the Pardee Chair of Computer Science.
At Berkeley, he led the design and implementation of RISC I, likely the first VLSI Reduced Instruction Set Computer. This research became the foundation of the SPARC architecture, currently used by Fujitsu, Sun, and Texas Instruments. He was also a leader of the Redundant Arrays of Inexpensive Disks (RAID) project, which led to high performance storage systems from many companies. These projects led to three distinguished dissertation awards from the Association for Computing Machinery (ACM). He is also co-author of five books and is chair of the Computing Research Association.
Patterson has won teaching awards from his campus, the ACM, and the Institute of Electrical and Electronic Engineers (IEEE). He is a Fellow of both the ACM and the IEEE, and is a member of the National Academy of Engineering. Reach him electronically at <patterson@cs.berkeley.edu> .

References

[Acha97] "Balancing Push and Pull for Data Broadcast," Swarup Acharya, Michael Franklin, and Stan Zdonik. ACM SIGMOD Intl. Conference on Management of Data. May 1997.

[Ande95] "A Case for NOW (Networks of Workstations)," T. Anderson, D. Culler, D. Patterson, and the NOW team. *IEEE Micro*, pages 54-64, February 1995.

[Apis96] "OC3MON: Flexible, Affordable, High Performance Statistics Collection," Joel Apisdorf, k claffy, Kevin Thompson, and Rick Wilder. *Proceedings of the 1996 LISA X Conference.*

[Case90] "A Simple Network Management Protocol (SNMP)," J. Case, M. Fedor, M. Schoffstall, and J. Davin. Available as RFC 1157 from http://www.internic.net/ds/dspg1intdoc.html

[Case96] "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)," J. Case, K. McCloghrie, M. Rose, S. Waldbusser. Available as RFC 1907.

[Cham76] "SEQUEL 2: A unified approach to data definition, manipulation, and control," Chamberlin, D. D., et al. *IBM J. Res. and Develop.* Nov. 1976, (also see errata in Jan. 1977 issue).

[Chan85] "Distributed snapshots: Determining global states of distributed systems," M. Chandy and L. Lamport. *ACM Transactions on Computer Systems*, February 1985.

[Chun97] "Virtual Network Transport Protocols for Myrinet," B. Chun, A. Mainwaring, D. Culler, *Proceedings of Hot Interconnects V*, Auguest 1997.

[Codd71] "A Data Base Sublanguage Founded on the Relational Calculus," Codd, E. *Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description*, Access and Control. San Diego, CA. Nov 1971.

[Deer90] "Multicast Routing in Datagram Internetworks and Extended LANs," Stephen E. Deering and David R. Cheriton. *ACM Transactions on Computer Systems*, May, 1990.

[Dolphin96] "HP Dolphin research project," Personal communication with author and some of the development group.

[Eswa76] "Specifications, Implementations, and Interactions of a Trigger Subsystem in an Integrated DataBase System," Eswaran, K. P. *IBM Research Report RJ1820.* August, 1976.

[Fink97] "Pulsar: An extensible tool for monitoring large Unix sites," Raphael A. Finkel, Accepted to *Software Practice and Experience.*

[Frei96] "The SSL Protocol: Version 3.0," Freier, A., Karlton, P., and Kocher, P. Internet draft available as http://home.netscape.com/eng/ssl3/ssl-toc.html .

[Gosl95] "The Java Language Environment: A White Paper," J. Gosling and H. McGilton, http://java.dimensionx.com/whitePaper/java-whitepaper-1.html.

[Gray75] "Granularity of Locks and Degrees of Consistency in a Shared DataBase," Jim Gray, et. al. IBM-RJ1654, Sept. 1975. Reprinted in Readings in *Database Systems*, 2nd edition.

[Hans93] "Automated System Monitoring and Notification With Swatch," Stephen E. Hansen & E. Todd Atkins, *Proceedings of the 1993 LISA VII Conference*.

[Hard92] "buzzerd: Automated Systems Monitoring with Notification in a Network Environment," Darren Hardy & Herb Morreale, *Proceedings of the 1992 LISA VI Conference*.

[HSV] "Hue, Saturation, and Value Color Model," http://loki.cs.gsu.edu/edcom/hypgraph/color/colorhs.htm.

[Hugh97] "Mini SQL 2.0," http://hughes.com.au/.

[Livn96] "Visual Exploration of Large Data Sets," Miron Livny, Raghu Ramakrishnan, and Jussi Myllymaki. In *Proceedings of the IS&T/SPIE Conference on Visual Data Exploration and Analysis*, January, 1996.

[Livn97] "DEVise: an Environment for Data Exploration and Visualization," Miron Livny, et. al. http://www.cs.wisc.edu/~devise/

[Mari97] "Marimba Castanet," http://www.marimba.com/

[Mill95] "Improved Algorithms for Synchronizing Computer Network Clocks," David Mills, *IEEE/ACM Transactions on Networking*, Vol. 3, No. 3, June, 1995.

[Murc84] "Physiological Principles for the Effective Use of Color," G. Murch, *IEEE CG&A*, Nov, 1984.

[Myri96] "The Myricom network," Described at http://www.myri.com/myrinet/

[Poin97] "PointCast: the desktop newscast," http://www.pointcast.com/

[Scha93] "A Practical Approach to NFS Response Time Monitoring," Gary Schaps and Peter Bishop, *Proceedings of the 1993 LISA VII Conference*.

[Sch93] "How to Keep Track of Your Network Configuration," J. Schönwälder & H. Langendörfer, *Proceedings of the 1993 LISA VII Conference*.

[Sch97] "Scotty Tnm Tcl Extension," Jürgen Schönwälder, http://wwwsnmp.cs.utwente.nl/~schoenw/scotty/.

[Seda95] "LACHESIS: A Tool for Benchmarking Internet Service Providers," Jeff Sedayao and Kotaro Akita, *Proceedings of the 1995 LISA IX Conference*.

[Ship91] "Monitoring Activity on a Large Unix Network with perl and Syslogd," Carl Shipley & Chingyow Wang, *Proceedings of the 1991 LISA V Conference*.

[Simo91] "System Resource Accounting on UNIX Systems," John Simonson. *Proceedings of the 1991 LISA V Conference*.

[SNM] "Sun Net Manager," Sun Solstice product.

[Sun86] "Remote Procedure Call Programming Guide," Sun Microsystems, Inc. Feb 1986.

[Wall96] "Perl 5: Practical Extraction and Report Language," Larry Wall, et al., Available from ftp://ftp.funet.fi/pub/languages/perl/CPAN/.

[Walt95] "Tracking Hardware Configurations in a Heterogeneous Network with syslogd." Rex Walters. *Proceedings of the 1995 LISA IX Conference*.

Monitoring Application Use with License Server Logs

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

One feature of our campus-wide UNIX service is the wide selection of scientific and engineering applications such as AutoCad, Pro/ENGINEER, Maple, etc. We currently have 32 "major application packages" site licensed, representing an annual cost of almost $300,000. A number of the licenses were based on concurrent usage, so around budget time, people started to ask if we had an appropriate number of licenses.

By adapting some previously developed software for tracking workstation use, we were able to determine who was using which applications, and concurrent usage information for these products and to reduce the number of concurrent users allowed to reflect actual use (plus some headroom). By applying these figures to just four applications, we were able to obtain a savings of $43,000 without cutting any service to our users.

This paper discusses the methods we used to collect, process, and display this information, as well as some of the problems we encountered.

Introduction

Three years ago, we presented a paper on monitoring workstation usage with a relational database [4]. This system worked by going to each machine and reading the `/var/adm/wtmp` files, and storing the contents in the database. When faced with the task of collecting application usage information, this system looked to provide a good starting point. We briefly examined at least one commercial product, but it did not take advantage of user demographic information – and the real show stopper at an educational site – it cost money.

While some of the applications we license are *node locked*,[1] a number of them rely on some sort of license server. One of our objectives is to have every application available on any of our machines. This way, rather than having to get a license for every machine, the license servers allow us to license a limited number of concurrent active copies of the application. As part of this process, the license servers often wrote log files with session information. While these log files were much different in format from the WTMP files, they were recording similar information. We were able to adapt our existing `Wtmp_Collector` program to pick up application usage information from these logs, and save them in database tables. This code was definitely worth keeping.

On the other hand, the original WTMP analysis program had some serious limitations; one of the big problems was the command line user interface. Each processing run required a number of parameters to be set, resulting in longer and longer command lines. Some common setting combinations were grouped together, but adding or changing them required recompiling the program. Even changing just one of the parameters would require rerunning the program, which repeated the database query step, obtaining the demographic information again. It was generally just a slow process, and it was just not ready to accommodate the demands placed on it to handle many different types of license server information. It also had to do much of the graphics processing. Although it was using jgraph [5] for actual postscript generation, there was still a lot of work involved, and there were some limits to what `jgraph`[2] could do for us. Given these limitations, as well as the need to accommodate changes in the data storage, we decided to replace the processing program. An X-based spreadsheet program we had already installed had an API that allowed us to load data directly into the spreadsheet, and build on the spreadsheet command menus. This also allowed us to use the graphics capability of the spreadsheet for output, as well as allow people to manipulate the data in a familiar spreadsheet environment.

With the updated data collector in place, and the new interface using the spreadsheet, we are now able to track the usage of many of our applications in addition to our workstations. This also simplifies the preparation of reports and graphs for management on a timely basis, without too much time or effort.

[1] A fun process where we have to supply a list of host ID numbers to the vendor, and they in turn return a list of magic numbers which we load into some file, and if the planets are aligned just right, things actually work. Doing this for hundreds of workstations is worthy of a paper in its own right.

[2] Jgraph is a program that reads a stream of simple drawing instructions, and generates PostScript on stdout. It works nicely in batch environments but it could not handle pie charts.

Taking License with Servers – Data Problems

The first step in the process of monitoring application use was to extract the raw data from the license server[3] log files and load it into the relational database tables. Although the information in the license server log files was in many ways similar to what we were getting from the WTMP logs, there were some significant differences as well. In addition, each type of log file had a different record format.

Database Record Format

One of the gains of this project was to put all license server records into a consistent format thereby simplifying later analysis and allowing us to compare usage of different applications. In determining the table layout, we started with the original WTMP table layout, redefined some columns, and added some new columns as seen in Table 1. We continued attempts to normalize some of the data at collection time, converting host names to host_IDs[4] and user names to owner ID[5].

[3]A note on the term *License Server*. From an operational standpoint, we often say that "machine xxx is a license server". That means that it is running one or more license server processes. However, in this paper, when we refer to a license server, we are generally referring to a specific process on a machine, providing licensing for one specific application or set of applications.

[4]A host_ID is an internal database key that identifies a particular host. Since all hosts of interest are already in the database, this key is easy to obtain.

[5]Like the host_ID, the owner_ID is a database key that uniquely identifies the user, even if the UNIX username is eventually re-used.

The first difference we had to accommodate between WTMP records and license server logs, was that while all of the WTMP records on a particular host applied to only that host, there may be more than one license server running on a given machine. In order to determine where in the particular log file it needed to resume processing, we needed to use both the `Lsrv_Host_ID` and `Application` columns to determine when we last collected records.

An additional complication was that some license servers provided license serving for more than one application; this required the addition of a subtype field in each record to differentiate between applications served by a particular license server process. To do this, we simply redefined the `Type` column, and for each license server (as defined in the `Application` column), would use one or more subtypes. Due to the number of different applications, both the `Application` and `Type` columns are required for the identification of a particular program product. This distinction would be significant for the data extraction process. It may have been preferable to make the `Type` column globally unique.

The last major change was the addition of a third host field in the record. In the wtmp records, we recorded from which host the record came, and for remote sessions, the host from which the user had connected to the target machine. With license servers, we also had to record the license server host in addition to the host on which the user was running the application, and in some cases, the actual location of the user.

Column	Type	Table	Notes
Username	char(8)	Both	Unix username.
Owner	Number	Both	Internal DB key for that username.
Host_ID	Number	Wtmp	Host_ID of the machine where the session took place.
Host_ID	Number	License	Host_ID of the user's machine, if available.
Lsrv_Host_ID	Number	License	Host_ID of the license server.
Application	char(4)	License	A short tag to identify which license server recorded the record.
On Time	Date	Both	The time and date when the session started.
Off Time	Date	Both	The time and date when the session terminated.
Line	char(12)	Wtmp	The TTY of the session, used to match disconnects with connects.
Line	char(12)	License	A tag to match session start with session end.
Type	char(1)	Wtmp	A code indicating the TYPE of session (telnet, ftp, etc.)
Type	char(1)	License	A code indicating subtypes for a license server.
Remote_Host	char(16)	Both	Machine where user originated the session if not the same as the host in host_ID above.
Rem_Host_ID	Number	Both	The host_ID of the remote host if it can be determined and is local.

Table 1: License Log Table Layout

Care and Feeding of License Server Logs

In order to provide a handle on the use of disk space by system log files, many of our log files are "rolled" on a periodic basis. That is, the existing file (named *filename*) is renamed to *filename*.1, *filename*.1 is renamed to *filename*.2, and so on for some number of generations[6] until the oldest one is deleted. This is normally done via cron and everyone is happy. As it turns out, many of the general UNIX log files are written via syslog, and fortunately, one of these "roll log file" cron jobs also `kill -HUP`s syslog so the new files are used. Other files, such as WTMP, are opened before each use, so the new file is used right away.

Unfortunately, the license servers generally do not write their log files via syslog, and so that while the log files might get rolled, the server keeps the file handle open and continues to write to the old file, leaving the newer files empty. To make matters worse, we found some cases of license servers that had been up long enough that the active file was rolled off the end and deleted. This resulted in no information being available, and the space being held by the still open log file until the license server was restarted or the machine rebooted.

This was not a good state of affairs. As a short term work around, we moved the roll code into the startup script for each license server, and set the number of generations to a moderately large number. We then scheduled a weekly reboot of each license server machine. This would ensure that each license server would get restarted (and the log files rolled) at least once a week. The collection process is still done on an occasional basis, and the large number of generations lets us get away with not collecting data frequently. The actual deletion of old log records is hopefully now being done by the collection process, rather than the log file rolling process.

Operation of the Collection Program

We periodically run the license server log collection program on all of our license serving machines. The general mode of operation is to scan all license server logs (and wtmp logs), and clean up when done. For testing, we generally run the program for just one specific type and without the clean option.

In scan mode, the collection program looks for a log file for each of the defined license servers and, if it finds one, checks in the database for the last time we collected records for that particular license server on that particular host. It then reads through the rolled log files until it finds the appropriate place to start. It next reads and processes the records, moving up through the rolled log files until it has read all the available records. If clean mode is also enabled, it will delete all but the most recent rolled log file. This leaves a little bit of a record behind on the machine, as well as the current active log file.

A structure built into the program stores the list of known license servers. This structure includes the application name for the license server (which is stored in the `Application` column of the database), the log file name, a time conversion routine, a parsing routine, and a list of valid subtypes (for translation into the `Type` column). Since many of the record formats are similar, this allows for a lot of code sharing between different log formats. While it might be preferable to store this information in a file instead of hard coding it, the parsing of the different formats is tricky enough that it is easier to hard code things than to encode the parsing information into a file.

Since the collection program deals with many different sources for the records, it is useful to convert the time and date into an easy to work with standard format. Internally, all date fields are converted to a standard form of seconds since 1 Jan 1970. Not counting the binary log file formats, we found four different date formats[7] that we had to handle.

Most of the logs are in a readable text format. You can read in a record, check the timestamp with the time conversion routine, and if you are interested in the record, call a sub-parse routine that returns pointers to username, remote host, record type, etc., which are then inserted into the database. The actual processing is based on the type of record. There is no standard for records types of course; each server uses its own keywords to indicate the type of record. So far, we have seen "OUT:", "IN:", "issued", "returned", "connect to", "closed connection to", and others along those lines.

The first record type is a session start. Before we insert a new record into the database, we check to see if there is a session open on that `Lsrv_Host_ID`, `Application` and `Line` combination; if there is, we terminate it on the assumption that we must have missed the session end record, since someone else is now using that same `Line`. This might happen in the case of an application that does not terminate cleanly, and so does not notify the license server on termination. We then insert a new session record into the database.

The second type of record is for a session end. For this case, the database is searched for a record for the same `Lsrv_Host_ID`, `Application` and `Line` that does not have an `Off_Time` set. Ideally there is only one, and the record is updated to reflect the end of the session. If we find more than one, we mark all of them as done, although we should set an error flag in the record.

[6] Actually, they are done in reverse order for obvious reasons.

[7] We wrote time conversion routines to handle: MM/DD HH:MI:SS, MM/DD HH:MI, MON DD HH:MI:SS (but no year!), and MM/DD/YY HH:MI:SS.

The third record type is a server restart. In this case, we go through and close all open records for that `Lsrv_Host_ID` and `Application`. The assumption here is that we will *not* get proper session end records, since the server state has most likely been lost.

Some license servers also generate time stamps. This is especially nice when you have backup servers running. For example, a server may be up for a long time, but may never be handling any requests since it is a backup server and the primary has stayed up. In such an instance, the timestamp records allow you to record that you at least checked the server.

Problem Data

We have run into some problems with attempting to process these log files. In one case, we were attempting to catch up on some old records (before we had implemented rolling for that particular license server), and discovered that the time stamps did not include the year, and we had three years' worth of data in the one file. The code that normally made a guess about the year ended up in a time warp. Fortunately, that was a minor service.

Some of the log files actually use multiple different formats in the file, and this can be very annoying when attempting to extract times or parse the fields. Other files are not in a text format. In some cases, such as the WTMP files, the record format is defined in a library. In other cases, such as the `NETLS` license server, the files are written in a proprietary database format, and instead of a record definition, they give you a program to extract data on demand. Although I was tempted to use this program, they had no facility to read an alternate database file, so all rolled information was lost.

Since we could not locate any documentation as to the file format, we had to reverse engineer the log file format. We were eventually able to figure out enough to eventually decode the records. A vendor supplied `struct` definition would have made things a lot easier.

Another source of error was with the assumption that keys we picked to identify session terminations were in fact unique. We had the case where one application was pretty slow to start up. Some users would double click on the application name, starting two copies. Sometimes they would even do it again, starting two more. Eventually, the first copy would appear, and then the second would appear right on top of the first. From the user point of view, they double clicked, and finally the application appeared and was operational. What the user did not know was that they had extra copies running in the background.)

We also had some license servers (or applications, we never determined who was not cooperating), that would not report the machine name on which the user was working. For example, if a user was using two workstations, or just had two sessions going, we faced the possibility of marking the wrong sessions as closed. We also had cases where the actual username and hostname that got logged was `anyuser@anyhost`; not very helpful.

Another potential source of problems comes from software upgrades. We have experienced problems with our WTMP logging after OS upgrades. On several occasions, we discovered that although we were successfully recording the start of a session, we were missing the end of a session, making it impossible to determine usage of labs. We could possibly see similar problems when we upgrade license servers.

In another case, we stopped getting session records from console users altogether, making it very difficult to determine if the workstation labs were being used. On reporting this to the vendor, we were told that it was not a bug, but rather that there was no interest in tracking console users. Finding this explanation unacceptable, we have since modified the login procedure to capture the information we need.

As we identified these various problems, we attempt to get help from the vendors in correcting them. We have also standardized our log file rolling script, and seem to be getting better at operating in a consistent manner.

Demographic Breakdowns or Breakdowns in Demographics

All of our user accounts are managed with a relational database, which can provide all sorts of demographic information about each userid, such as status (Student, Employee, Alumnus, etc.), departmental affiliation, campus address, and so forth. We are able to access this data when processing the log records from the WTMP database to provide additional information about the type of user.

We still faced some of the same problems with the demographic analysis, that we did with the original WTMP project. Specifically, while we do record the UNIX userid, the demographic information is obtained dynamically from the relational database, and that information is the current data, rather than the information that was in force at the time of the actual session. This results in what appears to be an increasing amount of use of our facilities by upperclassman and alumni as we examine older log records.

One approach to solving this problem is to add some additional fields. For example, when updating the demographic tables, rather than just making the change such as changing `Class_Year` from "freshman" to "sophomore", you could set an `End_Date` field in the current record to the current date and insert a new record with the `Start_Date` also set to the current date and the `Class_Year` field set to "sophomore". Similarly, when doing demographics lookups, instead of reading the latest demographic information, you select the record that matches the

appropriate time frame. In this way, you get the correct class year (or other desired demographic information) that applied at the time the session took place. This does however, involve changing the methods used to maintain the demographic data, and will cause the demographic tables to grow, rather then stay at a steady state size (since we will be adding records rather than just changing them.)

A second approach would be to record the demographic information of interest in the actual log records. This might be especially worthwhile if there a few fields that will be needed on a repeated basis, or that change frequently. Clearly however, some thought needs to go into the type of reports and breakdowns that will be needed. Another drawback to this approach is the delay between writing the record to the log file, and the eventual collection. If you are not collecting records on a timely basis, the demographics may change between the time of use and the time of collection.

Unlike the WTMP collection, which was confined just to hosts under central control, some of the licensed software is available for use by machines run by other departments. Unfortunately, there is no requirement that they use the same usernames that we use in the central systems. Although we can provide breakdowns of usage by machine and even by department, the user demographics are not valid for these "foreign" systems. We need to add a check in the collection program to determine if the user host is legal for demographic breakdowns or not, and include that status in record. Since the list of hosts where we can get valid demographics is subject to change as hosts are added to (or removed from) the central system over time, a `Demographics_Valid` flag needs to be set at data collection time.

Even if we are not able to determine valid demographic information on the users of a particular service, this data can still be of use. For one particular application, we were able to generate a pie chart showing usage by users in a particular department (based on host domain), versus the general population. With this, the administrators in the other department were able to justify paying for part of the cost of the application license, reducing the load on our software budget.

Programming to Xess – Talking to a Spreadsheet

One of the licensed applications we had for our UNIX workstations, was an X-based spreadsheet called `Xess`. One of the interesting features of this was an API toolkit [1] that allowed us to write a program that could connect to a running copy of the spreadsheet, add new menus and menu items, load data into the spreadsheet, and generate graphs.

Right out of the box, `Xess` looks like a normal spreadsheet, with tool bars and pull-down menus, and so on (see Figure 1). By opening the *Connections* menu, you can enable external connections[8] to the spreadsheet, which follow the X security rules. The external application is able to invoke many of the functions available to the console user, as well as insert data and formula into cells and read information back out. More importantly, it allowed us to define additional menus that could then invoke event-handling routines to perform the desired functions. This "dual control" feature was very useful during the development process, as we were able to have the program load some data into the spreadsheet, we could manually manipulate it, figure out pretty graphs and so on, and then encode the desired graphs and functions into one of our own menu items.

The data analysis generally consists of two phases, data selection followed by data processing. In the data selection phase, you need to specify the type or types of records you want, and the time period of interest. You may also want to specify additional conditions on the records. Once this is done, you then query the database for the desired records. As you can

[8]This can also be defaulted to **enabled** with an X resource setting.

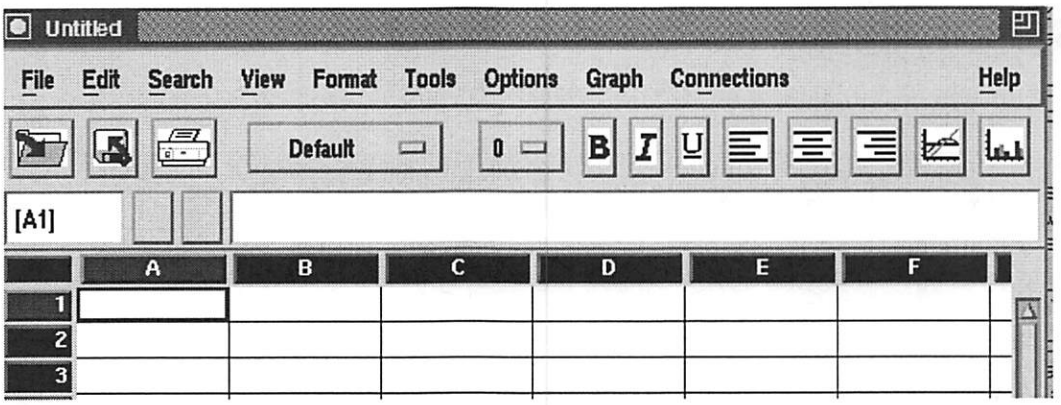

Figure 1: Default Xess Screen

see in Figure 2, we have added a number of additional menus to the spreadsheet session. Most of the selection phase is handled via the `Select` menu.

Once you have set the desired selection options, the last item you pick from the `Select` menu is the `Search Database` option, which extracts the data from the Oracle database. Once this is done, the `Select` menu is inactivated[9] and the `Process` menu is activated. At this point, you can select the type of data processing you want. The spreadsheet/menu interface made it easy to add more controls and options to the program. For the most part, the rest of the new menus just set flags and values in the program and modify the actions of the `Process` menu items. We frequently want to compare different demographic elements, and the `Column` and `Row` menus let us set what we put in the columns and what we put in the rows of the spreadsheet. We use the `DataType` menu to select what values we put into the spreadsheet (such as duration, number of sessions, etc.). A lot of different switch settings go into the `Options` menu. Unlike the previous three menus which act like radio buttons (selecting one deselects the others), the options are all set and cleared independently. During development, the `Debug` menu provided a handy place to put commands to provide debugging information, as well as to increase or decrease the verbosity level of the program.

The current version of Xess does not support cascading menus. This became a problem when dealing with some of the license servers that had a lot of sub-codes. We got around this in the `Select` menu, by prefixing each license server name with a *fraction* made up of the number of selected sub-codes over the total number of sub-codes for that particular server. The extraction program would also create (or rename) a new menu for that server, with each of the individual sub-code entries available for selection as desired. This approach might not be as elegant as cascading menus, but it works.

We frequently want to see usage as a function of the time of day, and often we want that to be an average for more than one day of use. To this end, we added a `Time In Columns` option to the `Datatype` menu. When this is selected, the columns of the spreadsheet each represent a specific time period. For example, in Table 2, we have few days of usage data.[10]

Although we could include the date with the time, and use more than 24 columns, we instead *wrap* the data at midnight and start the next day (we actually duplicate midnight at the other end). Once we have wrapped all of the data, we can simply insert a function to calculate an average load. In cases where there is a maximum number of concurrent sessions (such as a set number of licenses, or a fixed number of machines in a room for WTMP records) we were able

[9]Inactive menus and menu items are indicated by being grayed out. In Figure 2, the `Process` menu is currently inactive.

[10]This happens to be usage in a workstation lab, but the same processing applies to license server stats. Some data excluded for formatting reasons.

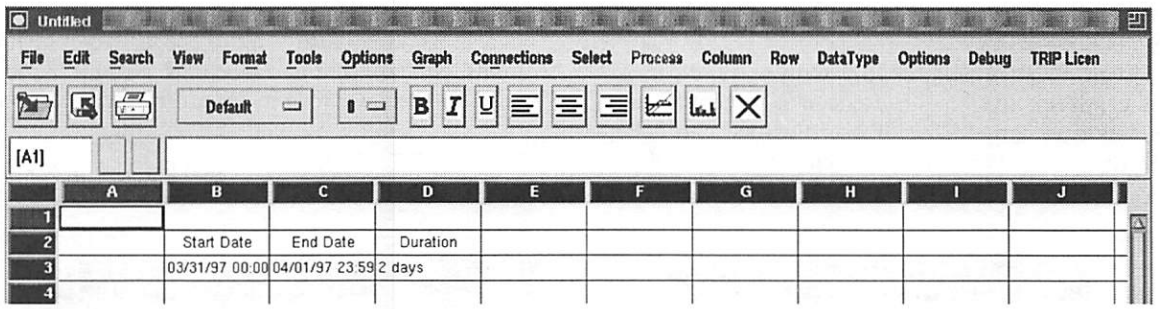

Figure 2: Enhanced Xess Screen

Date	00:00	01:00	02:00	03:00	04:00	05:00	06:00	07:00	. . .
12/02/96	5.471	1.480	0.096	0.300	0.001	0.000	0.024	1.524	
12/03/96	5.114	5.108	3.794	2.000	1.408	1.194	0.534	1.769	
12/04/96	5.816	2.973	3.604	2.053	1.056	1.000	1.200	1.768	
Mean(n=3)	4.995	3.848	3.537	2.794	1.741	1.244	1.124	1.336	
% Load	**71.4**	**55.0**	**50.5**	**39.9**	**24.9**	**17.8**	**16.1**	**19.1**	
Avg Load	0.714	0.550	0.505	0.399	0.249	0.178	0.161	0.191	

Table 2: Sample Spreadsheet Data.

figure a `Site Load`; the percentage of the resource in use, in this case, workstations. At this point, it is a trivial matter to have the spreadsheet generate a graph of average usage versus the time of day.

We often want to compare usage patterns over different time periods. To facilitate this, we added an option that, after the data is displayed, and the desired graphs are produced, it logically resets the origin of the spreadsheet (cell `A1`) to the row after the last active row, and then prompts for a new date range or search target. You can then select new data, load it into the spreadsheet, and generate new graphs for that selection. The program keeps track of the summary line of the past graphs, and combines them with the current summary, and generates a new graph with the different data sets displayed on the same axis. In Figure 3 for example, we have four weeks of data for two different labs, displayed on the same axis.

We added options to dump data in a raw form, and with selected demographic information added in case the researcher wants to do some other types of analysis. We were also able to dump session information to generate histograms of session duration, using the `histogram` graph option of Xess. Since all of the original functions of Xess are still available, after someone selects and processes data, does additional calculations, figures out what graphs they want and so on, they can then save the spreadsheet, using the options in the standard `file` menu, and allow others to look at the information at a later date. In fact, the data in Table 2 and graph in Figure 3 were taken from a run done months ago, and reloaded into Xess for this paper.

It's Easier the Second Time Around

In the original WTMP analysis, we were able to convert session data to load information, by creating a time-line, divided up into discrete segments of time, and "adding" in sessions to the appropriate bucket. Since this worked well, we decided to keep it. A distinct advantage we had in this rewrite is that we were building a new application from the ground up, yet we had a pretty clear idea of where we wanted to go, and what problems we had encountered in the first implementation. This helped us with designing data structures and the data flow through the program.

In the original program, it was sometimes difficult to break down the data by any arbitrary demographic value. There was a lot of special case code, and adding new types was tricky. In addition, we often wanted to break down records by two keys at once, which did not work at all well (you could break down in one direction, but not the other).

The new program needed to be much more flexible, and ideally, all data should be an abstraction. We also had a solid idea of what we were going to do with the data; specifically, dump it into a spreadsheet for display. This gave us three ways to break down the data; just provide a total in a single cell, which would be a zero axis case, break down by one demographic item, producing a single row or column, which would be the one axis case, or break it down by two demographic items, which would be a two axis case, essentially a mesh.[11] The determination as to how to divide the data, is controlled by the `Row` and `Column` menus.

As a record is read in, the row and column demographic elements are determined, and then they are compared to existing entries in a multiply-linked structure, and inserted or added to existing entries. In Table 3, we are looking at the number of sessions of Data Explorer, and breaking it down by the school affiliation in the rows, and the class year in the columns. Each entry is added in three times; once for

[11]The code to manage all of the data elements in the mesh was a good exercise in pointers and linked lists. I was able to successfully handle the three cases of n=0,1 or 2. I never tried to take it into more dimensions, although that might be possible.

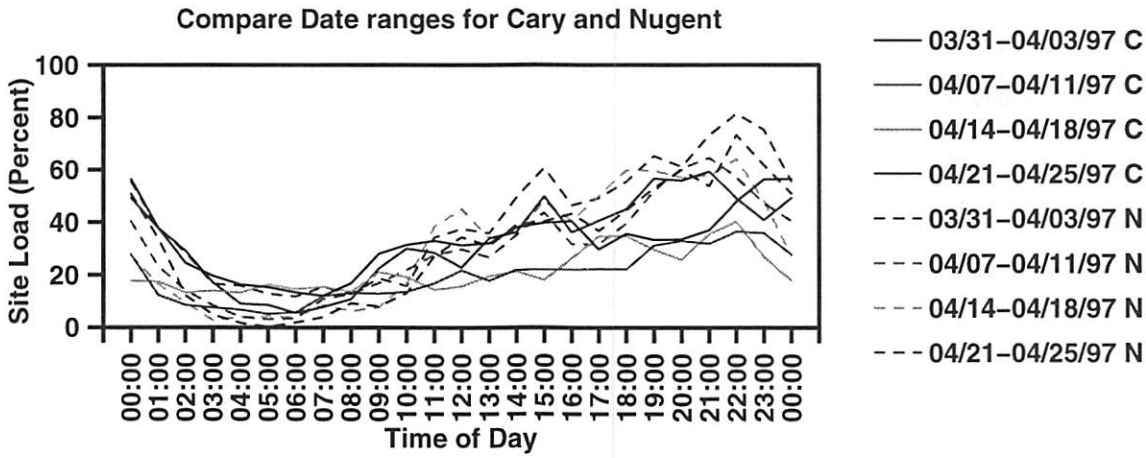

Figure 3: Combining Graphs on one Axis

the school, once for the class year, and once for that combination of school and class year. In this case, we can see the students in the school of science make the most use of the application, but that is mostly by Ph.D. candidates, whereas the Engineering students use it at the Masters and Undergraduate level as well.

When we query the database, we currently get back a linked list of database rows. This is due to our current use of our RSQL[12] interface to Oracle. When processing the selected data, each row is in turned loaded into a mesh data element, and passed along for processing into the mesh. Although this does increase memory usage, keeping the original query results in memory allows for very fast processing when changing demographic choices – all of the data is already extracted from the database. This is a big improvement over the old program, by caching all of the raw data and demographic information, we can readily play with the data and try different types of analysis without much of the delay we used to have.

Since this is one of the few places that we actually interact with the database, this routine provides a clean place to change the database API. While the RSQL is great for small queries, you do pay a performance penalty for larger queries. One area for future work is to replace this with a PRO*C[13] interface to allow block transfers of data. This can result in much better data transfer rates from the database.

If time-of-day processing is enabled, each mesh element also gets a pointer to an array of buckets, representing a time-line. This will hold session duration information that can later be extracted to produce the concurrent usage graphs shown earlier. I did not spend any time with changing the bucket size, although this could prove meaningful later on.

Space, the Final Frontier

In the original WTMP logging project, all WTMP records were stored in a single table. While this worked at first, after a year or so, this ever-growing table was becoming more difficult to manage. During system upgrades, we would have to export this single, multi-million record table to a single file, and finding single chunks of disk space big enough was becoming a definite challenge.

To work around this problem, we decided to break up the data into one-month sections, determined by the start date of the record. We now create a database table for each month, basically appending *_YY_MM* onto the table name. At collection time, insertions are easy; you just take the connect time and generate the table name. Session terminations are a bit trickier, as a long session may have started in the previous month, or even earlier. We therefore currently look back two months for a session start record.

The change for the analysis program was also moderately simple. Instead of just taking the date range and returning a table name, we return a list of table names. The selection process then queries each table in turn, returning a set of record for each month. These are then processed into the mesh in turn. The only challenge remaining is to be sure to have the new tables created ahead of time so there is a place to receive new data.

In addition, we also had started systematic backups of the Simon database [2, 3], and the WTMP data became the bulk of the data being backed up. The Oracle database maintains a record of all transactions since the last full backup. These transaction files multiplied quickly with the log collection, requiring more frequent full backups, and a lot more free disk space on the database machine to hold transaction files between backups.

The Simon database is used to manage all of our UNIX accounts, disk accounting, and many other aspects of our operation. Since the operational need for the WTMP and license server data was much less than the rest of the Simon data, we decided to move the WTMP data into a different database instance that was not being backed up. This removed the load on the backup system, while still allowing us to back up the mission critical Simon data.

Although the log record tables could continue to grow in size, in practice, we have not always reloaded the older data when moving the database between machines. Also, as the data ages, the problems with demographics get worse.

[12]We developed a layer that runs on top of the Oracle Call Interface that simplifies writing applications that need to interface with Oracle, it handles network authentication, space management and many other details of coding with Oracle.

[13]PRO*C is the Oracle pre-compiler for embedding SQL queries (and other calls to the database) directly in a C program.

Sum		DOC	MAS	SR
	Cat Value	540331	54698	53852
Sci	460870	460354	0	516
Eng	188011	79977	54698	53336

Table 3: Data Explorer use by School and Class.

Findings and Losses

Although we had continued to collect WTMP data, until the license server collection project had started to heat up, no one had really spent much time doing analysis of the WTMP data. When we started planning for some workstation lab upgrades, we went to generate some nice reports with the new tools and discovered that as a result of an OS upgrade, we did not have accurate termination information for most of our public workstations labs. As a fallback position, we did some analysis on just the session start times and we were able to draw some conclusions on workstation usage.

Despite the problems with the workstation usage statistics, we were able to use them in some decision making process. In addition, the very concrete benefits from the license server analysis has brought a good deal of attention and support to the statistic collection and analysis project. Additional staff have been assigned to work with the system on an ongoing basis, and session data collection will be integrated into our existing session management software.

The information collected allowed us to measure the actual usage of a number of our licensed applications, and in some cases, we could reduce the number of copies licensed, resulting in significant savings in license fees. This savings should help maintain interest and support for the project. We also need to find ways to measure the usage of applications that do not use this type of license server.

This actually opens up other questions about measuring application usage. Given that this project grew out of measuring workstation use, essentially seat time, most of the information has a bias to session duration. This makes a lot of sense for an interactive program such as autocad, but makes less sense for an application that might remain open but unused for most of a session such as a mail reader. Recording session duration is just about meaningless for applications such as compilers and text processors; the fact that someone spends a lot of time running latex may be more a reflection on the speed of their machine and not on the type of work they are doing.

References and Availability

All source code for the Simon system is available for anonymous FTP. See `ftp://ftp.rpi.edu/pub/its-release/simon/README.simon` for details. In addition, all of the Oracle table definitions are available at `http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html`.

Although this system benefits from being able to extract demographic information from Simon, and from actually storing the raw data in the database, both of these areas are pretty well isolated, so other databases, or even flat files could be used in place of the connections to Simon.

Xess is a commercial spreadsheet product available for many UNIX platforms and WindowsNT. More information on Xess can be obtained from:

Applied Information Systems, Inc.
100 Europa Drive Chapel Hill, NC 27514 USA
1-919-942-7801
1-800-334-5510
1-919-493-7563 FAX
info@ais.com.
http://www.ais.com

Acknowledgments

Many thanks to Adam S. Moskowitz and Debra Wentorf for their help in editing several drafts of this paper (although I still take full responsibility for all misplaced commas and poor grammar) and also to Jackie Blendell for her help with the original abstract. Thanks also to my manager, Kathy Bursese, who encouraged preparation and submission of the paper.

Author Information

Jon Finke graduated from Rensselaer in 1983, where he had provided microcomputer support and communications programming, with a BS-ECSE. He continued as a full time staff member in the computer center. From PC communications, he moved into mainframe communications and networking, and then on to Unix support, including a stint in the Nysernet Network Information Center. A charter member of the Workstation Support Group he took over printing development and support and later inherited the Simon project, which has been his primary focus for the past six years. He is currently a Senior Systems Programmer in the Server Support Services department at Rensselaer, where he continues integrating Simon with the rest of the Institute information systems, and also deals with information security concerns. Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via http://www.rpi.edu/~finkej.

Bibliography

[1] Applied Information Systems Inc, Chapel Hill, NC. *Xess Connections API Toolkit Guide – Version 3.1*, 1996. http://www.ais.com.

[2] Jon Finke. Automated userid management. In *Proceedings of Community Workshop '92*, Troy, NY, June 1992. Rensselear Polytechnic Institute. Paper 3-5.

[3] Jon Finke. Relational database + automated sysadmin = simon. Boston, MA, July 1993. Sun Users Group. Invited Talk for SUG-East 93.

[4] Jon Finke. Monitoring usage of workstations with a relational database. In *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pages 149-158. Rensselear Polytechnic

Institute, USENIX, September 1994. San Diego, CA.

[5] James S. Plank. Jgraph – a filter for plotting graphs in postscript. In *USENIX Technical Conference Proceedings*, pages 61-66. Princeton University, USENIX, January 1993. San Diego, CA.

Automating 24x7 Support Response To Telephone Requests

Peter Scott – Jet Propulsion Laboratory, California Institute of Technology

ABSTRACT

Demands for uninterrupted availability of systems that were hitherto not viewed as critical to the success of the enterprise has placed a burden upon support infrastructures for those systems. The small-to-medium help desk for a system that comprises part of an enterprise information system is under increasing pressure to provide round-the-clock support, while staffing budgets may not have caught up with the fiscal realities of covering second- or third-shift on-site personnel. While the volume of calls outside normal hours may be small, a guaranteed turnaround and response to emergencies is essential for many IT operations to gain the trust of their customers.

This paper describes a system for automated answering of the help desk telephone during non-peak hours, and for notifying on-call staff of emergencies within minutes. The system uses two voice-capable modems on a well-maintained computer for presenting to the caller a typical phone menu hierarchical menu which may include an option to record a message about some type of emergency. In this event, the system will notify the staff who are on call at that time for that type of emergency, contacting them by means they have specified in advance, such as voice telephone, numeric or alphanumeric pager, or other means. The system is currently undergoing testing in the Enterprise Information System File Service at JPL.

Motivation

After years of preaching the advantages of current technologies such as distributed systems, management acceptance comes with the price of making those technologies ubiquitous, easy-to-use, and reliable. The Enterprise Information System (EIS) Project at JPL was established in 1996 in an ambitious plan to establish a unified and coherent strategy and implementation of computing infrastructure to serve the laboratory's reengineering plans. Early adoption and roll-out of AFS (a distributed filesystem from Transarc), and a current migration path to DCE/DFS are some of the components of that implementation. Making this technology accessible and even transparent to a laboratory population highly diverse in its computer expertise and extremely heterogeneous in its platform usage demands a capable and responsive help mechanism.

The File Service element of EIS, having rolled out its technology fairly soon, set up an e-mail alias and a mnemonic telephone number for support. The nature of space missions and international collaboration means that JPL is open 24 hours a day, every day of the year. If flight projects whose personnel might work during off-hours were to embrace the new technology (a key aim for the reduction of overall costs and achievement of stretch goals), these projects and their personnel needed to know that their emergencies during off-hours would receive prompt attention. However, the start-up nature of the File Service meant that the number of customers was too small to justify funding for continuous on-site assistance.

In order to break this Catch-22, it was determined that on-call support would suffice, since response time could be fairly rapid. This led to a "pass the pager" scheme for the on-call person of the week (or month). This has a number of disadvantages:

- Only one person can be on call (the phone system will not forward a number to more than one pager at a time).
- Switching to another person requires that they meet in the same location at the same time as the desired handoff.
- If the pager is not working or out of range, the user does not know that their call has not been received.
- Granularity of on-call scheduling is extremely coarse, on the order of one day. If someone were able to answer calls from 6pm to 2am but no later, and someone else were able to answer calls from 2am to 6am but no earlier, for instance, one or both of them would have to compromise.
- If the support person's schedule changes after they leave work, handing off to another person can be difficult.
- Completeness verification and reporting of the on-call schedule is decoupled from the page forwarding itself and therefore subject to inaccuracy.

It is not acceptable to require customers to memorize multiple numbers, or use different numbers at different times, or look up the current phone number over the web (since the very reason they may be calling is to report that the network is down!).

Solution

What is needed is a mechanism for connecting the calls coming in to a single number with the multiple different and time-dependent methods of contacting on-call staff. A large enough organization could afford to staff the helpline with a person around the clock who need know no more than who to reach for a particular problem at any given time, and how. Some organizations cannot accommodate the budget necessary to pay for second- and third-shift personnel, and automation suggested itself in this case.

We proposed and constructed a system for automated answering of the help desk telephone during non-peak hours, and for notifying on-call staff of emergencies within minutes or seconds. No on-site personnel are needed provided that the on-call staff at any time are within a suitable distance for emergencies that cannot be resolved from wherever they are at the time. The scheduling of the on-call staff can be done by the staff themselves, and reports generated from the on-call schedule highlight any gaps in coverage. Nothing in the design of the system is inherently specific to the particular project it serves.

Requirements

The requirements we imposed on the system were as follows (KEY: **R**: Required for initial system; **RF**: Required for some future system; **D**: Desirable for initial system; **DF**: Desirable for some future system):

Schedule

- **R** – Maintains a user-editable schedule of who is on call at what times.
- **D** – User-friendly and quick to use interface for specifying when you are on call.
- **R** – Records incoming calls and makes outgoing calls to people on the schedule.
- **R** – System allows for more than one person to be on call at a time.
- **R** – System can provide a map of coverage showing who is on call for what reasons during specified upcoming period.
- **RF** – System allows users to be on-call at different times for different reasons.
- **RF** – System allows users to specify different actions to be taken when they are called for different reasons.
- **RF** – System supports multiple schedules specifying different sets of people to be called in response to caller selecting different options.

Input

- **D** – System will take a message even if caller does not press any touch tones.
- **R** – Ability to hook into programs that monitor system health and automatically call when malfunction detected.
- **D** – System can be triggered by e-mail.
- **DF** – System has API for triggering from Perl scripts.

Output Methods

- **R** – User can specify how they are to be contacted: voice phone, numeric pager, alphanumeric pager, e-mail.
- **R** – System can make multiple outgoing calls to different numbers and devices to contact the same person.
- **DF** – System can take alternate action in the event that a phone number on outgoing call does not answer.
- **D** – System allows user to specify multiple methods for contacting them.
- **R** – System allows custom audio to be played according to user specification in schedule.
- **D** – System allows users to record their own custom audio.

Paging – Alphanumeric

- **R** – User can specify the display message in the case of alphanumeric pager.
- **RF** – System can make outgoing calls to human-operated paging service.

Paging – Voice

- **DF** – System allows user to specify and/or record audio to be played to human-operated paging service.
- **RF** – A user-selectable option when calling out is to play the incoming message to the answerer.
- **DF** – A user-selectable option is to play only the first N seconds of the incoming message to the answerer.

E-Mail

- **D** – System can send e-mail as part of a response cycle.
- **D** – E-mail part of response is configurable by user.
- **D** – System can send e-mail for every request, configurable by administrator, in addition to any e-mail sent out to people on call.

Menu System

- **D** – Menu system provides a tree which can be traversed with touch tones, providing information and taking calls.
- **D** – Menu output can be preempted by caller pressing a valid touch tone.
- **RF** – System allows reason for call to be a function of where in voice menu system caller selected an option to record message.

Commercial Products

The first time we conducted a search for COTS products to perform this task, we were unable to find any costing under $100,000 - we did not investigate systems costing over $100,000 which might have had these capabilities. We found several inexpensive packages offering phone menu capabilities for personal computers, but not providing notification to voice, pager, and email according to a schedule.

A more recent search conducted for this paper turned up the Tel Alert product from Telamon, Inc., [6] which performs the phone menu functions in addition to status monitoring and contacting by pager, and has some scheduling capability; although it requires proprietary hardware it is worth evaluation before implementing an in-house solution. VoiceGuide from Katalina Technologies [7] performs most required functions (without the scheduling sophistication), plus many others (such as caller ID detection), at a very low price, but on Windows only. It could be used to call out to an external program which implemented the scheduling algorithm (perhaps remotely on a Unix box), although if the helpline application requires a high degree of robustness, most people would not choose to run it on a Windows machine.

Among freeware products, tpage [5] allows scheduling of people to be paged but not with the same flexibility and readability that this system provides, and only contacts pagers, not voice phones.

Construction

The key to the system is a voice-capable modem such as the ZyXEL Omni 288. A daemon waits for an incoming call and when one arrives, presents a typical hierarchical phone menu to the caller ("Press 1 if you know the NAME of..."). In fact, the daemon is simply playing a series of sound files that have been placed in a particular directory and given names corresponding to the keys the user should press to select them. Subdirectories containing more sound files are used to represent deeper levels of the menu, and the

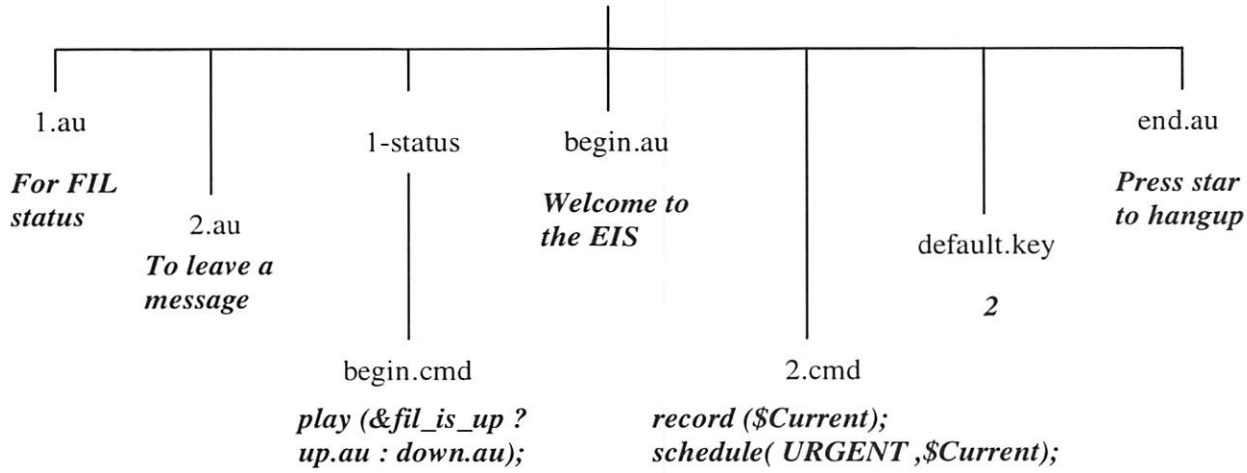

Figure 1: Menu directory hierarchy.

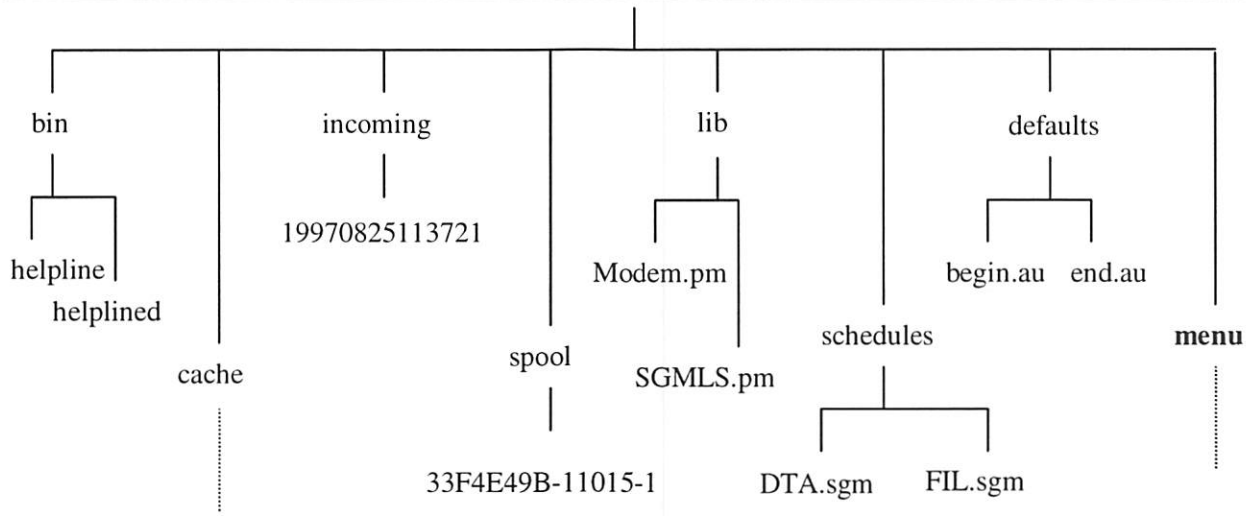

Figure 2: Tool hierarchy.

traversal follows some simple rules for checking the existence of files or directories with names matching particular expressions.

The directory hierarchy of the this menu looks something like Figure 1. The algorithm for processing this tree is very simple; see Figure 3. The directory tree for the phone menu is part of the hierarchy for the tool, which looks something like Figure 2 (not all files displayed).

An advantage of this approach to constructing a phone menu system is that it is so easy to create additional levels that an entire support staff can be given write access, allowing them to create new sound files and menu levels if they wish. This distributes the job of maintaining the phone menu system among the people with the domain expertise appropriate to each particular part of the hierarchy.

In addition, responses can be generated on the fly according to information derived at call time. The menu can call scripts that can execute arbitrary Perl commands, for instance, to ascertain the status of various institutional servers, and modify the playback content accordingly. The daemon runs each script with a Perl `eval` command so that a subroutine library is imported into the namespace of the script enabling it to perform certain phone menu-specific operations

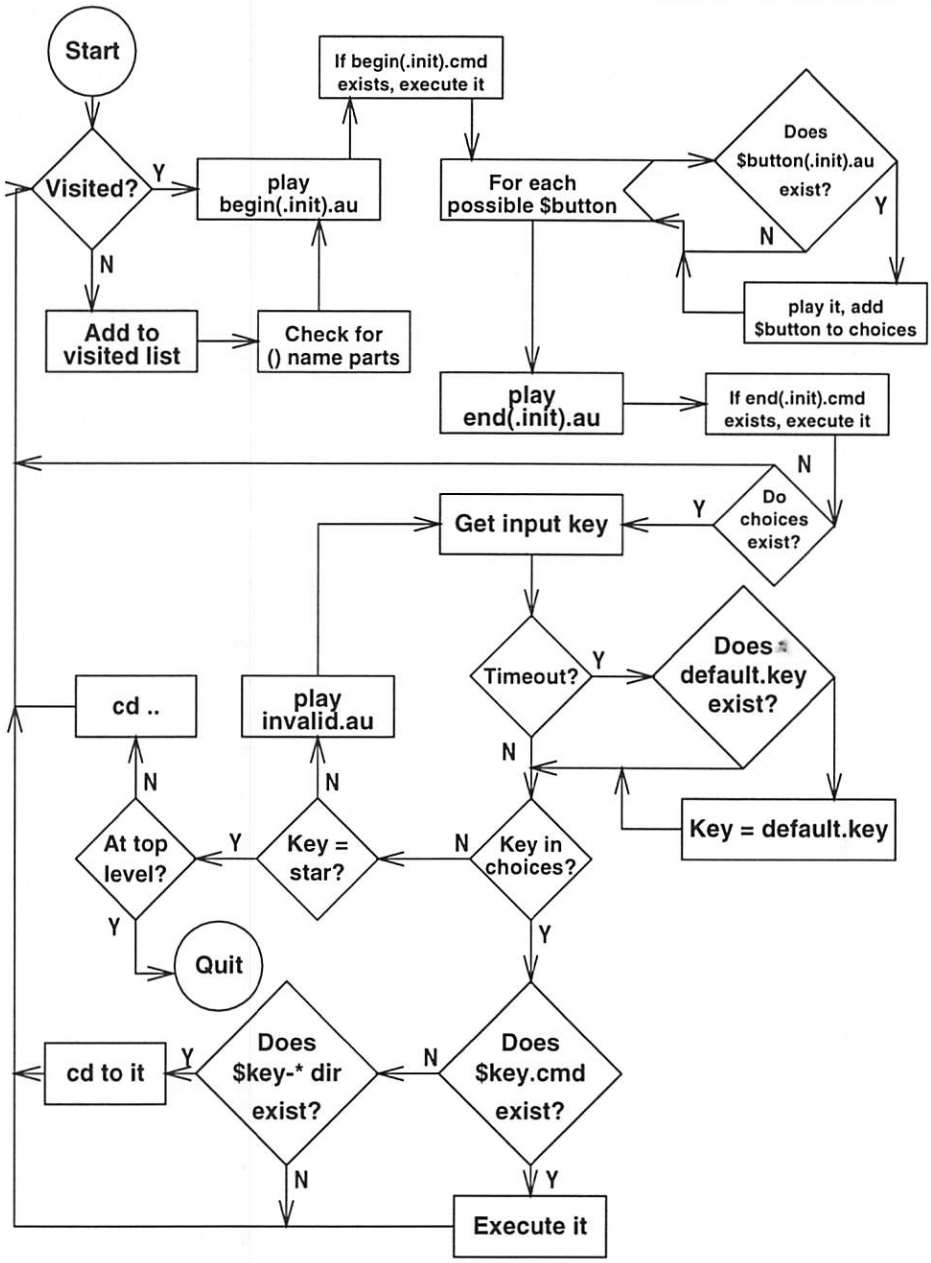

Figure 3: Tree processing algorithm.

easily. The key point here is not that a phone menu system can respond with dynamic information, but that the design of this implementation allows it to be done so easily using only standard tools (simple Perl, and a microphone hooked up to a recording program).

One operation a caller may wish to perform is recording a call for help. The phone menu interface for handling this is very simple: at the node where the corresponding sound file says, "If you are experiencing trouble with..." the associated script contains:

```
$record_file = nextname();
record ($record_file);
schedule (<situation>,
    $record_file, "FIL");
```

```
<Day Name="Weekdays">
 <DayRange>
  <DayFrom>   <WeekDay Day="Mondays">  </DayFrom>
  <DayTo>   <WeekDay Day="Fridays">  </DayTo>
 </DayRange>
</Day>
<Day Name="Holidays">
 <DayRange>  <OneDay Name="NewYearsDay"> </DayRange>
 <DayRange>  <OneDay Name="MemorialDay"> </DayRange>
 <DayRange>  <OneDay Name="IndependenceHoliday"> </DayRange>
 <DayRange>  <OneDay Name="Thanksgiving"> </DayRange>
 <DayRange>  <OneDay Name="Christmas"> </DayRange>
</Day>
<Day Name="WorkWeek">
 <DayRange>  <OneDay Name="Weekdays"> </DayRange>
 <Except>  <OneDay Name="Holidays"> </Except>
</Day>

<Situations>
 <Situation Type="AFSDown">
 <Situation Type="WWWDown">
 <Situation Type="Urgent">
</Situations>

<Profile Name="George1" Priority="CallOnlyIfAlone">
 <Command Number="44321" Type="Voice">  <PlayFile>
 </Command>
 <Condition Type="DoOnFailure">
 <Command Number="97907560" Type="Voice">
  <PlayFile Name="~george/audio/sounds/pageme-intro.au">
  <PlayFile>
 </Command>
</Profile>

<Schedule>
 <ProfileName Name="Glenn1">
 <Situation="AFSDown"> <Situation="Urgent">
 <DateTimeRange AllDay="Yes">   <OneDay Name="GlennWork">
 </DateTimeRange>
 <DateTimeRange>  <OneDay Name="WeekEnds">
  <TimeFrom Hour="8">  <TimeTo Hour="18" Minute="30">
 </DateTimeRange>
 <DateTimeRange>  <OneDay Name="GlennHolidays">
  <TimeFrom Hour="9">  <TimeTo Hour="17">
 </DateTimeRange>
</Schedule>
```

Listing 1: Scheduling examples.

where <situation> is a mnemonic for the type of emergency being reported (any number of situations is possible). The first line acquires a unique name for the incoming sound file, the second guides the caller through a phone menu recording menu which allows them to review, cancel, or send a message, and the third notifies the scheduler of an occurrence of that particular type of emergency.

The spool directory contains files queued for processing by `helplined`, the daemon which handles outbound communications. A spool file looks like:

```
SITUATION: AFSDOWN
PROFILE: PETER
CONDITION: NULLCONDITION
COMMAND_TYPE: VOICE
COMMAND_NUMBER: 42246
COMMAND_PIN:
COMMAND_ADDRESS:
COMMAND_PLAYFILES: ~peter/audio/
    sounds/attention.au ..
    ./incoming/19970825113721
COMMAND_MESSAGE:
```

On-call Scheduling

The on-call notifier and scheduler is an independent component which could be implemented entirely separately from the phone menu system; it is not necessary for emergency notifications to come from the phone menu system, since any process which executes the schedule() subroutine can make such a notification. Thus it is possible to cause on-call notification in the event of automatically-detected or triggered events such as a UPS alarm signalling power failure, a network health monitor alarm, or SNMP events.

When an emergency notification occurs, the scheduler parses a schedule file to see who is on-call at that time for that type of situation. The schedule file contains statements describing four entities:

- **Date Ranges** - assigning identifiers to a set of dates;
- **Situations** - a list of valid situation identifiers for the schedule;
- **Profiles** - assigning identifiers to methods of reaching a particular person;
- **Schedules** - associating certain situations with certain profiles during specified times over certain date ranges

When it has determined which profiles are active for the given situation at the current time, the scheduler runs the profile to contact the on-call support person by pager, telephone, or whatever other method is specified in their profile.

The schedule file is written in SGML; a DTD was created to describe schedules. This provides two advantages: firstly, that the input and editing of the file can be done through an SGML editor which guides the user through the creation of a syntactically valid entry, and can be customized to provide a high degree of user-friendliness; thus, entries can be made by the on-call support personnel themselves. Secondly, that enforcing of the required syntax by the SGML editor leaves almost no syntax checking required of the application. Whenever an emergency notification occurs, the schedule file is read through the nSGMLS [3] parser with the SGMLS.pm module [2] and rejected if a syntax error is found (of course, the file is also validated every time it is edited).

The SGML schedule in Listing 1 contains examples of each of the four statements listed above. The date range definitions of the individual holidays and the personal date ranges have been elided to save space.

The SGML is not designed for human parsing nor editing, and fortunately, neither is necessary. There exist a number of products which read the DTD and provided structured editing of the source, guaranteeing that the syntax is valid. A freeware example is the SGML module for EMACS. Some Commercial Off-The-Shelf (COTS) products also allow "style sheets" to be constructed which hide the tags completely from the user, providing WYSIWYG editing like modern HTML editors (e.g., Arbortext's Guided Editor). When formatted for human-readable output, the statements in the schedule can be more easily illustrated in some more examples:

Date Ranges

```
DAY ChristmasEve: 12/24
DAY Christmas: FROM ChristmasEve
        TO BoxingDay
DAY BobWork: WorkWeek EXCEPT
        BobVacation
```

The first example assigns a specific date (which can be qualified with a year if necessary) to the identifier `ChristmasEve`, the second example assigns a range of already-assigned dates to another identifier, and the third assigns a range of dates excluding a predefined subrange. The first two lines would have been input by an administrator to define certain well-known dates; since the year is they will not need to change fixed-date definitions such as Independence Day from year to year. The third line would have been input by Bob, a member of the support staff, defining a date range corresponding to his default work week: the administrator-defined workweek (weekdays except for holidays), minus his personal vacation (defined elsewhere).

Profiles

```
PROFILE Bob1: VOICE 5557458
   (MESSAGE      ~bob/sounds/
   oncall1.au, $current),
IF FAILED PAGE 5553453 (PIN
    3894634)
PROFILE Jack3: ONLYIFALONE
```

```
IXOPAGE 5550532
(MESSAGE "I've fallen and I
   can't get up")
PROFILE Jim2: PAGE 5550974 (PIN
      63459023)
   THEN EMAIL jim@tarkus (MESSAGE
      "It's dead, Jim")
```

The first example tries to call Bob at his home, playing first an introductory sound file recorded by Bob which can both explain things to his wife if she answers, and is timed to go past the outgoing message on his answering machine if they're not in, and then plays the message let by the caller; if there is no answer at that number, it calls Bob's numeric pager. The second example calls Jack on his alphanumeric pager, but only if he's the only one on call at the time. The third example calls Jim on his numeric pager, and then, regardless of the success or failure of the page, sends him an email message.

Schedules

```
SCHEDULE Bob1, Jim2 FOR PWRDOWN
    20:00 - 05:00 DURING WorkWeek
SCHEDULE Jack3 FOR NETDOWN 09:00 -
    18:00, 20:00 - 22:00 DURING
    Weekends
SCHEDULE John FOR ANYTHING 07:00
    - 01:00 DURING AllWeek
```

The first example attaches the profiles `Bob1` and `Jim2` to a late-night/early morning slot during the work week (previously defined as being `AllWeek EXCEPT Holidays`, where `Holidays` was defined as the union of the defined holidays), in the case of power failures. The second example attaches the profile `Jack3` to two time slots on weekends for network failures. The third example attaches John's profile to a punishing schedule in case of any situation.

We can now do two things with this schedule file: we can produce a report showing the on-call

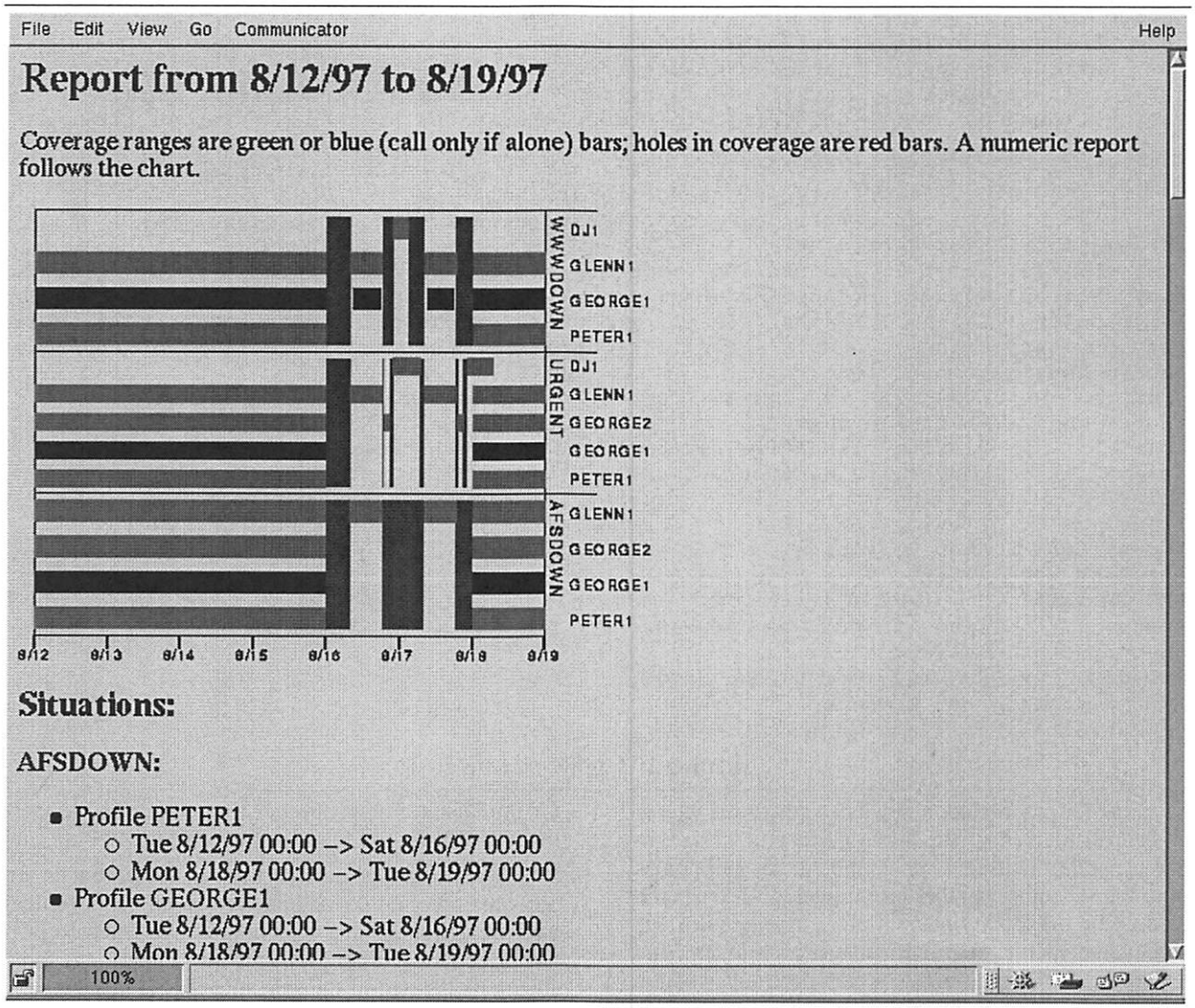

Figure 4: Web Page.

```
:             Toggling DTR to reset modem
:             Configuring port parameters.. (/dev/cua/a / 38400 bps)
:             Initializing Modem
Modem Cmd:    Command Mode?                    ATE1V1 -> OK
configure:    Modem was in commmand mode.  Now in command mode.
Modem Cmd:    Initialization String            AT&FS2=255S0=0S13.2=1N0&H4 -> OK
Modem Cmd:    Switch to VOICE mode             AT+FCLASS=8 -> OK
Modem Cmd:    Default settings for voice mod AT+VIP -> OK
Modem Cmd:    Turn up DTMF sensitivity         AT+VDD=3,8 -> OK
Modem Cmd:    Enable XON/XOFF Flow Control     AT+FLO=1 -> OK
:             Modem Initialized
Modem Cmd:    Ping                             AT -> OK
main_loop:    Waiting for a call..
main_loop:    MODEM: [
main_loop:    Waiting for a call..
main_loop:    RING detected!
Modem Cmd:    Answering VOICE call             AT+VLS=2 -> VCON
handle_call:  Call received.
handle_call:  Voice::init complete.
menu:         Menu /afs/jpl/group/ets/src/helpline/menu
say_choices:  Using .../cache//afs/jpl/group/ets/src/helpline/menu/begin.init.au.zyxel
Modem Cmd:    Start Playing                    AT+VTX -> CONNECT
play:         Playing voice file .../cache/.../menu/begin.init.au.zyxel
play:         Transmission Buffer Underrun
Modem Cmd:    Stop Playing                     <DLE><ETX> -> OK
say_choices:  Running script begin.init.cmd
say_choices:  Using .../cache//afs/jpl/group/ets/src/helpline/menu/1.au.zyxel
say_choices:  Using .../cache//afs/jpl/group/ets/src/helpline/menu/2.au.zyxel
say_choices:  Using .../cache//afs/jpl/group/ets/src/helpline/menu/star.au.zyxel
Modem Cmd:    Start Playing                    AT+VTX -> CONNECT
play:         .../cache//afs/jpl/group/ets/src/helpline/menu/1.au.zyxel
              .../cache//afs/jpl/group/ets/src/helpline/menu/2.au
              .../cache//afs/jpl/group/ets/src/helpline/ menu/star.au.zyxel
Modem Cmd:    Stop Recording                   <DLE>! -> OK
say_choices:  Using .../cache//afs/jpl/group/ets/src/helpline/default/record/1.au.zyxel
say_choices:  Using .../cache//afs/jpl/group/ets/src/helpline/default/record/2.au.zyxel
say_choices:  Using .../cache//afs/jpl/group/ets/src/helpline/default/record/3.au.zyxel
say_choices:  Using .../cache//afs/jpl/group/ets/src/helpline/default/record/pound.au.zyxel
Modem Cmd:    Command Mode?                    ATE1V1 -> OK
play:         Modem was in commmand mode.  Now in command mode.
Modem Cmd:    Start Playing                    AT+VTX -> CONNECT
play:         Playing voice file .../cache/.../default/record/1.au.zyxel
play:         Playing voice file .../cache/.../default/record/2.au.zyxel
play:         Playing voice file .../cache/.../default/record/3.au.zyxel
play:         Playing voice file .../cache/.../default/record/pound.au.zyxel
Modem Cmd:    Stop Playing                     <DLE><ETX> -> OK
process_keys  Waiting 5 seconds..
MODEM:wait:         DTMF digit: "#"
menu:         Ready to process keypress: #
menu:         Valid choice.
process_keys  Running script pound.cmd
run_script:   send message
33F4E5A6-11173-1 ( PETER )
```

Listing 2: Logfile excerpts.

coverage for any period, and we can determine who is on call for a given situation at a given time. The second capability is used when an emergency call comes in or the scheduler is otherwise triggered; the first can be used by the facility manager for planning purposes and to demonstrate to senior management proven 24x7 coverage. The same module which determines which profiles are active at the current moment has routines for producing a report between two dates and rendering it in HTML; see Figure 4 for example.

The horizontal bars on the `GEORGE1` tracks are blue; the other horizontal bars are green. The vertical bars are red, indicating gaps in coverage when no profile is active. This report shows that there are three situations defined in the schedule file, labelled `AFS-DOWN`, `WWWDOWN`, and `URGENT`.

Implementation Details

The system is implemented using custom Perl 5 code for answering the phone and calling out. We experimented heavily with vgetty from Gert Doering's mgetty+sendfax package [1] and liked the scripting capabilities, but ultimately were confined by problems with data synchronization and capabilities for extension in the version we had at the time. Our system nevertheless suffers from synchronization problems where sometimes keypresses are not detected or the modem goes into an unintended state, but we believe these are tractable problems. The schedule parsing is done with custom Perl 5 code using David Megginsons's SGMLS.pm SGML-parsing module [2] and James Clark's nsgmls SGML parser [3]. Turning the schedule file into a report or a list of profiles on-call at the time was made much easier with Perl 5's object-oriented features. Where sound files do not exist for messages to be played to the caller, voice synthesis is done with rsynth [4], by Nick Ing-Simmons (usually used for error messages).

A logging capability allows the operator to view the progress of the system. Parts of a log of a session recording a call for help are reproduced in Listing 2.

Experience in Use

The most difficult part was communicating with the modem; we have not yet figured out how to handle data overruns or why touch tones are sometimes missed. We heard at the Perl Conference that a module for modem control is in the works and that may solve this problem. The tests for failure of an outgoing call are incomplete; we would prefer more sophisticated checking for whether a call was successful.

Conclusions

The helpline system demonstrates that the tools exist to construct an apparently complex system in a domain hitherto considered the province of COTS-ware using freeware components. The use of SGML to specify the grammar of an input file provides a significant gain in user-friendly editing and syntax checking.

Acknowledgments

Josh Wilmes, of RPI, developed the modem-answering code while working as a co-op student at JPL.

The work described was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

Author Information

Peter Scott left Cambridge University with an MA in CS in 1983 and has been working at the Jet Propulsion Laboratory in Pasadena, California, since 1986. For the last two years he has been lead development engineer on the File Service Element of the Enterprise Information System. He can be reached at M/S 301-342, Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109 and at <pjscott@euclid.JPL.NASA.GOV>.

References

[1] mgetty+sendfax: http://www.leo.org/~doering/mgetty/index.html.

[2] SGMLS.pm: http://www.uottawa.ca/~dmeggins/SGMLSpm/sgmlspm.html.

[3] NSGMLS: An SGML System Conforming to International Standard ISO 8879 – Standard Generalized Markup Language. http://www.jclark.com/sp/nsgmls.htm.

[4] Rsynth: ftp://svr-ftp.eng.cam.ac.uk/comp.speech/sources/rsynth-2.0.tar.gz.

[5] tpage: ftp://gumby.dsd.trw.com/pub/tpage.tar.Z.

[6] Tel Alert: http://www.telamon.com/prodinfo/telalert.html.

[7] VoiceGuide: http://www.ozemail.com.au/~katalina/vgidx.htm.

Turning the Corner: Upgrading Yourself from "System Clerk" to "System Advocate"

Tom Limoncelli – Lucent Bell Labs – Murray Hill, NJ

ABSTRACT

Two (of the many) types of System Administrators are identified: The "System Clerk" simply performs clerical tasks as she is told to do them (creating accounts, installing software) while the System Advocate proactively works with users to develop and evolve the environment.

Transitioning from Clerk to Advocate involves a change of attitude and "making time" for advocacy. Ways to develop more "spare" time for pro-active work are also spotlighted. I also explain ways to avoid the "system administrator visibility paradox": nobody notices the system administrator on days when things are working well.

Introduction

Here's a test for system administrators: Do users walk into your office with software to install that they procured, licensed, and had shipped to them? When a new employee is hired, are you asked to create her login after she has arrived for her first day? Does your salary come from the same budget as the clerks and the mail-room, not a technical budget?

If you said "yes" to any of these questions, it may be a warning sign that you are being treated as clerical overhead rather than as part of the main business processes of your company. This paper explains how someone can make the transition from System Clerk to a proactive role that I call the System Advocate. Making this transition is not easy. Advocacy requires having "spare" time which usually means both automating processes that are currently performed manually and avoiding time-wasters. This transition is key to improving the service you provide to your users. As a result, it can help your career or at least keep you from being bored.

Changing a habit is one of the most difficult things a person can do. This paper is about changing many, many habits. If you choose to make this transition, you will discover that it is not easy. One should be realistic and expect the changes to happen slowly and be uncomfortable at first. Eventually you will see the rewards. Once the new, good habits have replaced the old, it will be difficult to break them!

What Is Clerical?

If someone hand-writes a memo, it can be given to any clerk/typist to be typed. Clerical SA tasks should be automated and documented (e.g., creating accounts) or simple enough (e.g., installing packaged software[1]) that anyone can do them.

[1]When clerks run into problems installing software they call the toll-free number listed in the manual instead of reverse engineering the install script and proceeding manually.

You can be more valuable to your company and enhance your career by making the move from System Clerk to System Advocate. The value to the company is obvious: by being part of the software purchasing process, you knew to purchase additional disk capacity at the same time as the new software. Result: Your users were not waiting extra weeks while the order for additional disks goes through purchasing. By being part of the new employee orientation planning, you were informed in advance that she was coming and can ensure her account was created and ready when she arrived. Result: the new employee has a better impression of your company (which is important, she took a big risk to move here). She feels welcomed because everything was ready for her when she arrived.

Your career is enhanced because you are providing better value to your company, and a fair and ethical company rewards that. You also feel better about yourself because you are now a part of your environment, not doing tasks that are considered "afterthoughts." You are also less likely to be bored because less of your time will be spent on menial tasks.

What Is A System Advocate?

A System Advocate is a system administrator who completes clerical tasks quickly (usually through automation) so she can focus on finding ways to make her users more productive and advocating for the users' needs to management. How do you know what makes users more productive? Ask them. Be ready with ideas, but let them take the lead.

Example 1:

A user says she could get more done if her computer was faster. You watch her work for a while and hear her swap disk "going nuts." After verifying the problem with `pstat` and other tools, you conclude her machine needs more RAM. A day later you walk

into her boss's office with the user and a quote for more RAM. You've educated the user so she can state her case, but you are there if she stumbles or if the boss has technical questions. The boss agrees to purchase the RAM. Soon you have a more productive user and a feather in your cap.

Example 2:

A programming shop would be more productive if it had debugging tools like PureAtria's Purify or CenterLine's CodeCenter. You get sales literature from both companies and bring them to lunch with the lead programmer. Soon you are arranging sales presentations and evaluations with both companies. You and the lead programmer work up a quote for the software, disk space, etc. and present it to your boss together. She buys it, the programmers are more productive and happy: you all win!

A System Advocate is pro-active, a partner of the users who can help sell ideas to management. The clerical tasks have been automated and become a minor part of your brain-time.

The System Advocate Attitude

An advocate has "the right attitude." I'm surprised how often I have to remind system administrators that their users are not "lusers" or "pests with requests." They are the reasons you have a job. You are there to serve, and more importantly to collaborate, and advocate. You are part of the team.

If you mentally think of them as "customers"[2] whom you serve, instead of "pests" that make requests all day, your attitude can change dramatically. However, a System Advocate can get into trouble if one adopts the attitude that "the customer is always right." You could end up doing people's jobs for them if you do not remember to help users help themselves. It is a balancing act.

Requests from users are opportunities to do a fantastic job, one you can be proud of. When you integrate this attitude into your habits, your users will notice the difference in the service you provide!

When I receive email reporting a problem, my reply now ends with a sincere note of thanks for reporting the problem and that "it helped me fix the problem and look for ways to prevent it in the future." If a user makes a request often, I look for ways to empower the user to help themself (maybe `sudo` or other `suid` binary that lets them do the task) or to find a more permanent solution.

Sometimes the solution is as simple as keeping the spare toner cartridges by the printer so users can change them if you aren't around. If a user accidentally deletes files often and you waste a lot of time each week restoring files from tape, you can invest time in helping the user learn about "rm -i" or use other "safe delete" programs. Or, maybe it would be appropriate to advocate for the purchase of a system that lets users do their own restores. If you maintain a log of the number and frequency of restore requests, management can make a more informed decision. (or decide to talk to certain users about being more careful.)

"Spare" Time

You can't make the transition without investing time. "But I have zero free time!", you say. Then first you must invest time in automating tasks so that later you will have more "spare" time.

Creating Spare Time

It takes a long time to manually create a user account. Investing two hours to automate this task will have a dramatic payoff. Much of the reason you save time will be that you will no longer spend time fixing mistakes that are inevitable with manual processes. These mistakes make you look sloppy, even if they only happen once in a long time.

Automation opens doors to better ways of doing things. Without automation, you might choose to put off all account creation requests until a certain day of the week, then do them all in "batch." Users see extremely poor turn-around time. If the automation makes creating accounts simple, you can do them "on demand."

The more you automate a process the better service you can provide. I worked at a site that did OS loads and upgrades manually. Each time they had to "from memory" remember what links to make, files to change, etc. This was followed by 2-3 days of fixing the inevitable mistakes as users reported problems. The result was a network of machines that never changed their OS after the initial load and never had security patches or bug-fixes loaded because it was too much work. Another site had a manual process (requiring you to lug a CD-ROM player to the machine) but loading/upgrading the operating system was one of two simple procedures followed by running a script that did everything else. In one evening an admin could upgrade 15 machines and new machines could be loaded in an hour. Within three months of a SunOS release the entire network was upgraded. Currently I work at a site that uses Sun's AutoInstall for Solaris. By typing "`boot net - install`" at the console, a machine boots from a `bootp` server, loads/reloads its OS from scratch, installs our favorite patches, and installs our local modifications; lately we don't even stick around to make sure the procedure worked because it always does. At night machines check for new patches in our patch library, load them, and reboot if required. Our entire network upgrades itself nightly. Users now have machines ready to use 30 minutes after we've

[2] "Whither The Customer?" Kevin C. Smallwood, ;login: and counterpoint by Rob Kolstad

unpacked it and all machines in our system have consistent environments.

To find potential areas of improvement, it may help to keep a log of what you do each day for a week. Either write down what you do as you do it, or have your computer beep every 15 minutes as a signal to log whatever you are doing right now. At the end of a week, look for tasks to automate. When automating a task, do not get bogged down in creating the ultimate system. A simple script that assists you with "the common case" may be more valuable than a large system that automates every possible aspect of a task. The script I use to configure our `bootp` server to accept a new NCD X-terminal simply outputs the commands I should type to create the proper symbolic links. If I approve, I cut and paste them to a shell prompt. The script was easy to write because it doesn't have to deal with asking, "Are you sure?" or special cases. If the NCD requires something special I can use the output as the basis for the commands I actually type.

Finding "Hidden" Spare Time

Where do you find the spare time to create the automation that will give you more spare time? Spare time is hiding all over. Look at these places for some:

- Most businesses have a "light" time of the year. Software shops with a new release every four months usually have a 3-4 week "slow period" after each release. You might have spare time then, or this may be the busy time for you as you get things ready for the users' busy period. You may find that during their busy period nobody bothers you.
- Stop reading Usenet. Period.
- Remove yourself from the two busiest mailing lists you are on. (But make sure you are subscribed to `cert-announce` of course!)
- Take advantage of mail filtering software such as `procmail`[3] to put mail from mailing lists into one folder which is read weekly.
- Take advantage of the early morning: Come in an hour earlier and handle many small, daily tasks when nobody is around to interrupt. Don't waste this time cleaning your email box!
- Have your boss send you to a 1-day time management class. Painful as they sound, they do teach some amazing time-saving tricks. I feel like I have gained an extra two hours each day when I use the techniques I learned.
- Invest in training workshops or books that will enable you to automate tasks (write code) in less time. If you don't know `perl` you are working too hard. Don't overlook `make` either.
- Hold weekly or monthly meetings with your chief customer (a manager or department head) to set priorities and eliminate superfluous items.
- Hire an assistant. If your boss reads this paper and agrees with the philosophy, she may agree to getting you a part-time assistant to take care of some of the clerical tasks. Possible candidates: A local high school student, a technically inclined secretary, the person that covers for you when you are away, a summer intern, an interested programmer. Local students are particularly useful because they are cheap and are looking for ways to gain experience.

Now you have the spare time you need to automate the boring, error prone stuff that eats your time. Remember: "Investing time in automating tasks doesn't cost, it pays."

A Single Success

Now you have some "spare" time, pick one situation to "advocate." Start with something small that, if successful, will be visible. Your goal is to help a user be so successful that she tells others. The best advertising is word of mouth. This also means not biting off more than you can chew. Bust your butt to make sure you follow through and make this a success for the user.

Now repeat. You may have to do this a number of times before you create a success that gets the visibility you hoped for.

Finding places to be helpful can be difficult. I recommend eating lunch with different users every day and keeping your eyes open for opportunity. Avoid the temptation to say "yes" to everyone. Your first attempt requires your undivided attention. Don't say "yes" to to more than one simultaneously task until you've got a feeling for the work involved.

Nobody likes an overbearing System Advocate. Warning signs to look for include: you dominate the conversation, you are loud, you always have something to say no matter what topic comes up, you try to "one-up" other people's anecdotes with better ones.

Your Personal Positive Visibility

The System Administrator's Visibility Paradox is that you are only noticed if something breaks. Six months of 100% uptime takes a huge amount of behind the scenes work and dedication. Management may get the impression that you aren't needed because they do not see the work required. Then a server crashes every four hours until a controller is replaced. Suddenly you are valuable. You are the hero. You are important.

This is not a good position to be in because users will get the impression that you are doing nothing 95% of the time because they don't see the important, behind-the-scenes work you do.

[3] `procmail` by S. R. van den Berg, <ftp://ftp.informatik.rwth-aachen.de/pub/packages/procmail>

A bad alternative is to not maintain a stable system so you are always needed. Bad idea.

A better idea is to make sure people know you do a lot in the background to keep things running smoothly, but do it without being overbearing. Remember:

> YOU are responsible for your own
> Personal Positive Visibility!
> No one does it for you!

How can you do that? Here are three techniques that not only improve your personal positive visibility, but are good pro-active things to do anyway!

Technique #1: Be Your Best

None of the other techniques will work if you aren't providing good service to your users.

Technique #2: The System Status Web Page

Everyone loves surfing the web. During a recent major network reorganization at Bell Labs we maintained a web page with an easy-to-type URL that listed current status:

> SYSTEM STATUS
> No known problems at this time.
> Mon Jul 2 10:20:13 EDT 1997
> Please report problems via email to "help"
>
> Coming soon:
> - Wed, July 10: 10am-noon
  - Aisle 4D5 scheduled power outage
> - Thu, July 11: 12:15pm
  - Router reboot (2 minutes)

> Changes recently completed:
> - Fri, June 28:
  - System "londo" and "gkar" upgrades complete
> - Fri, June 30:
  - System "elbows" converted to flamenet

You might configure users' browsers to go to this page when users click on the "Home" button and provide a number of useful URLs at the top of the page so there is less temptation to change the default.

At times the system status was changed to simply, "Machine narn is down, we're working on it." This says to users "we care" and "we are working on the problem." In the absence of information users often assume you are out to lunch, ignoring the problem, or ignorant of the problem.

Obviously you care about the problem and are working on it, but taking 10 seconds to let your users know this creates the Personal Positive Visibility you desire.

As users become accustomed to checking this URL before complaining, the number of phone calls you get while trying to fix the problem is reduced. There is nothing worse that being delayed from working on a problem because you are busy answering the phone calls of users that want to "help" by reporting the problem. Users will not develop the habit of checking this URL until you consistently update it.

You might also consider putting a chalkboard at the entrance to your machine room and putting status messages there. Low tech, but it works.

Technique #3: User Meetings

Another technique is to host regularly scheduled meetings with users: Users' Group meetings can be an excellent forum for bidirectional feedback. They can also be a disaster if you are not diplomatic. Be careful. The focus must be on their needs, not yours.

I suggest you use a format similar to this:

Welcome (2 minutes) – Welcome the group and thank them for attending. It is a good idea to have the agenda written on a white board so people know what to expect.

Introductions (5 minutes) – Each attendee introduces themself.

User Feedback (20 minutes) – Getting feedback from the users is part art, part science. You want users to focus on their needs. You may consider taking a class on meeting facilitation if you feel weak in this area. What works for me is to ask open-ended questions like:

"If one thing could be improved it would be . . ."

"The single best part of our computing environment is . . ."

"The single worst part of our computing environment is . . ."

"My job would be easier if . . ."

Keep a log of what users suggest. A huge pad of paper on an easel is best so everyone can see what is being recorded. Don't write full sentences, just key phrases like "faster installation new C++" or "add CPUs to server5." Once a page is filled, tape it to the wall and go to the next question.

Do not reject or explain away any requests. Just record what people say. To get the best responses, people must feel they are safe. People do not feel safe–and will stop talking–if every suggestion is answered with your reason why that "just isn't possible" or "we can't afford it." However, be clear that recording an idea does not guarantee it will be implemented.

Be careful of permitting one talkative person to dominate the meeting. If this happens, you might want to go around the room having each person answer the question in sequence or use phrases like, "Let's hear from the people in the room that have not yet spoken."

Don't get stuck in an infinite loop. If you hit your time limit, cut off discussion politely and move

on. People have busy schedules and need to get back to work.

Review (10 minutes) – Review what you've recorded by reading the items out loud. Consider reviewing these lists with your boss afterwards to prioritize (and reject) tasks.

Show and Tell (30 minutes) – This is the main attraction. People want to be entertained and the best way to entertain technical people is to have them learn something new. Ask a salesperson to present information on a product, have an internal person present some information about something they do or found useful, or you yourself might want to focus on a new feature of your network. This may be a good time to explain a big change that is coming soon, or describe your network topology or some aspect of the system that you find users don't understand.

Meeting Review (5 minutes) – Have each person in turn say how they felt the meeting went (less than one sentence each).

Thank everyone (2 minutes) – First ask for a show of hands to indicate if they thought this meeting was useful. Then remind people that you are available if people want to drop by and discuss these issues further. Then (and this is important) thank people for spending time out of their busy schedule.

Your Boss

It is a good idea to show your boss this article before you begin. A good boss will support the gradual change to System Advocate because she understands how this will create value to your company. A really good boss will help you set the pace, make suggestions, and mentor you. Warning: a good boss will set a pace that is slower than you want. Trust her advice.

A bad boss will say this paper is trash and you'll be forced to use these techniques independently until you have made dramatic improvements in your users productivity.

Conclusion

Transforming a clerical role to a pro-active "System Advocacy" role is a change in attitude and working style that can dramatically improve the service you provide to your users. It isn't easy. It requires hard work, and requires investing time to "create" free time. Achieving your first success can be very difficult. Taking ownership of your Personal Positive Visibility not only enhances your career, but provides opportunities to serve your users better. To lower cost of ownership, vendors are automating the clerical side of system administration but no vendor can automate the value of a System Advocate.

References

Love Your Job!, By Dr. Paul Powers & Deborah Russell, 1st Edition August 1993, O'Reilly and Associates.

The Macintosh Way, By Guy Kawasaki, September 1989, Scott Foresman Trade.

Programming Perl, 2nd Edition, Larry Wall, Tom Christiansen and Randal L. Schwartz, 1996, O'Reilly and Associates.

System Performance Tuning, By Mike Loukides, 1st Edition November 1990, O'Reilly and Associates.

When You Can't Find Your UNIX System Administrator, By Linda Mui, 1st Edition April 1995, O'Reilly and Associates.

"Whither The Customer?" Kevin C. Smallwood, *;login:* and counterpoint by Rob Kolstad

Author Information

Tom Limoncelli is a MTS at Bell Labs, the R&D unit of Lucent Technologies, where he is chiefly concerned with the architecture and operation of the data network for much of Research. Tom started doing system administration on VAX/VMS systems in 1987 and switched to Unix in 1991. He holds a B. A. in C. S. from Drew University, Madison, New Jersey. His homepage is http://www.bell-labs.com/user/tal; he can be reached at <tal@lucent.com>.

How to Control and Manage Change in a Commercial Data Center Without Losing Your Mind

Sally J. Howden and Frank B. Northrup – Distributed Computing Consultants, Inc.

ABSTRACT

Most computer system problems today are caused by change. Change is an innate characteristic of an active computer system. This paper presents an approach whose goal is to minimize and control the impact of problems by controlling and managing change. It is geared towards the system administrator's role in meeting this goal in a commercial data center environment.

System administrators have been given the task of providing reliable, available and supportable computing environments for their clients. A system which does not meet these requirements results in, at the very least, lost productivity, but may also cause a financial loss, and in the worst cases may result in injury to the customers which the business serves (see [1] for an example). In order to provide a reliable, available and supportable computing system it is necessary to minimize the impact on the system's users when problems occur. Almost all computer system problems today are caused by change: changes in hardware components; changes in system or application software; and to a lesser extent changes in processes and/or procedures, or in personnel. The extent to which a system administrator is able to control and manage change is the extent to which they are able to provide a reliable, available and supportable computing system to their client(s).

This paper describes a platform independent approach for pro-actively managing problems in a computing system by managing change well. This approach includes: the process of documenting the computer system's current state; the process of documenting the change; and the process and conditions under which the change is first implemented in a test environment, then in a pre-production environment and finally in a production environment. This approach saves time and effort in the long-term administration of computer systems. The documentation necessary to facilitate this approach is described and some examples provided. This approach is currently being used by Distributed Computing Consultants, Inc. (DCCI) with its clients.

Introduction

Almost all computer system problems today are caused by change. A computer system is defined as a collection of people, hardware, software, and documentation that provides a particular service to customers. Hardware components of a computer system tend to fail during one of two times: the initial *burn-in* period – shortly after having been added to the system; or after having been in operation for greater than five years. However, in today's rapidly evolving computing environment hardware is often replaced with new and improved components well before five years pass. Furthermore, the failure of a hardware component may have little or no effect on users due to the increasing number of vendors who provide systems with redundant components such as CPUs, power supplies and fans; disks with automatic failover capabilities; and hot swappable disks (coupled with mirroring or other RAID approaches). Thus, the hardware failure which is most likely to impact users is that which occurs when a change is made to the hardware.

Likewise, the majority of software failures occur (or in the case of bugs are discovered) when the software is first used following initial installation, after a version update or in a way which is different than originally intended. Again, a problem occurs when a change is made.

When a problem or failure occurs within a computing environment, the first question an experienced system administrator will ask is: *What changed?* Was something added? Was something modified? Is someone trying to use the system in a way different from the defined use? Has something in the computing system's environment changed? Was there a recent power outage? Is there a problem with the environmental controls? This course of investigation – figuring out what *change* has occurred – will often lead the system administrator to the cause of the failure.

So, if change *is* the problem in many cases, a logical course of action would be to minimize the number and length of disruptions to the computer system caused by change. Why should a system administrator be concerned with pro-actively managing

changes to a computer system? Why not just react to change by *fixing* the problems as they occur?

Today, computer processes are integral to almost every aspect of a company's business. An outage can impact product research, development, marketing, time to market, accounting, finances, regulations, payroll, etc. In several of these areas, any outage directly impacts a company's bottom line and ability to compete effectively in the open market.

In a commercial data center every day matters. In pharmaceutical companies, from the time the company submits an application for a new drug to the United States Federal Drug Administration (FDA) it has seven years during which it owns exclusive rights to the drug. However, the company can not sell the drug in the United States until the FDA approves it. After the seven year period, other companies are free to copy and market the drug. The time between the FDA approval and the end of the seven year period can be highly profitable for the pharmaceutical company (i.e., hundreds of thousands of dollars *per day*). Therefore, any computer outage that lengthens the time between application to and approval by the FDA has significant detrimental impact.

Now suppose the company has several drugs at this same stage when the outage occurs.

And the financial well-being of a company is not the only concern. With hospitals, insurance companies, doctors' offices, pharmaceutical companies and pharmacies the loss of information access can directly impact a person's life.

In many ways universities are very similar to commercial entities. A key product for most universities is education. At many universities major changes are scheduled during breaks. The university's ability

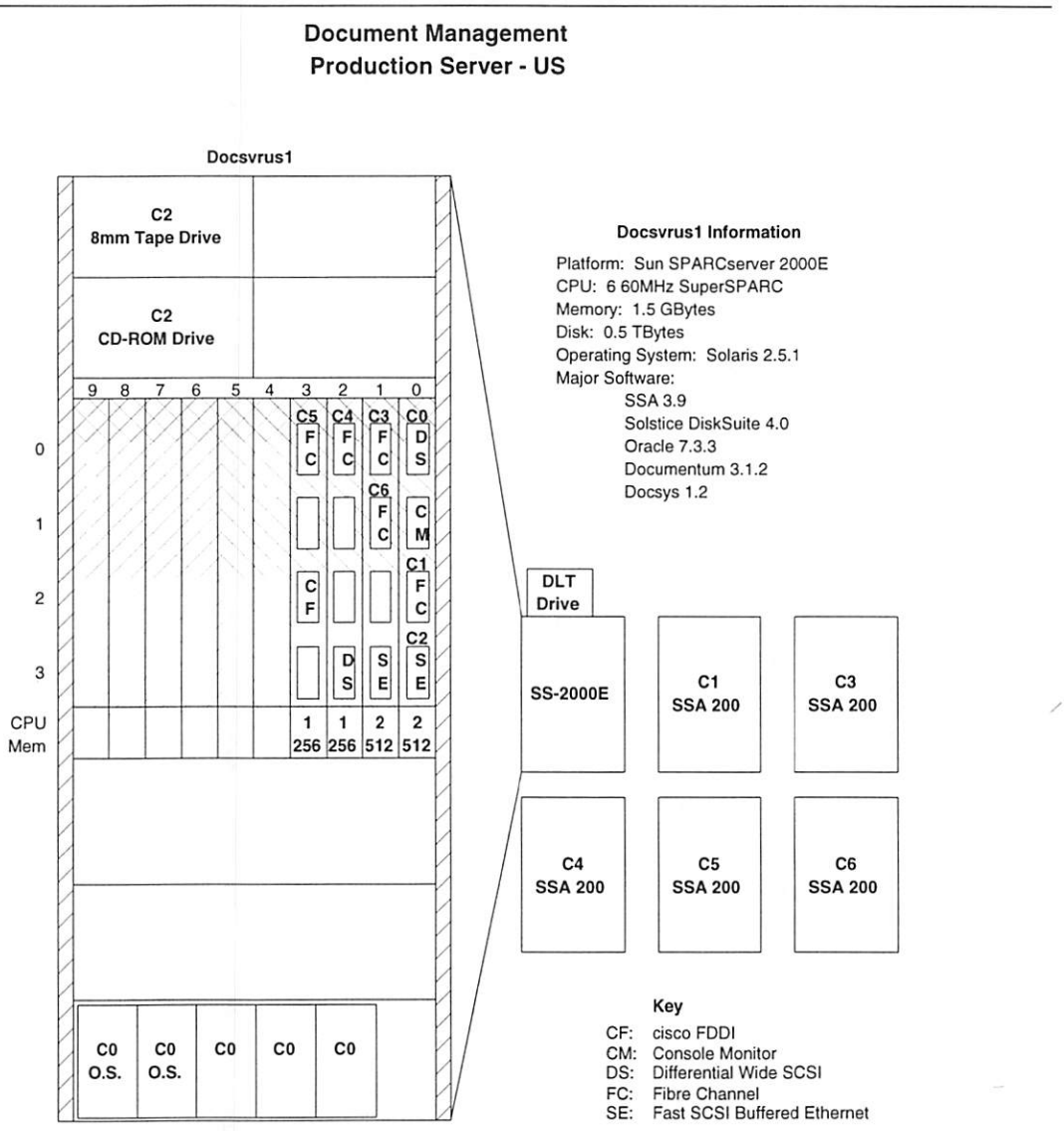

Figure 1: Example of a Server Configuration.

to deliver its product to market would be greatly hampered if its computer system was unavailable during the term.

So, if an unavailable computer system can have such a drastic affect on the client, it is vitally important to the client that the computer process not be interrupted. For this reason it is important that the system administrator do as much as possible to insure that the computer process is not interrupted.

The truth is, even a seemingly insignificant change can cause a problem which requires valuable time (including a system outage) to correct. This does not imply that we should never make changes to production systems. Today, change is the rule, not the exception. Therefore system administrators need to think carefully about where, when and how changes are performed. The rest of this paper presents one approach for proactively managing problems by managing change well throughout the life of a computer system [2] rather than reactively dealing with changes as they occur. This approach advocates the following [3]:

1. Establishing & documenting the system's baseline.
2. Understanding the change.
3. Testing the change on test and pre-production systems.
4. Documenting the change before, during and after implementation.
5. Reviewing the change with peers, manager and client before, during and after implementation.
6. Defining a back-out strategy for the change.
7. Training all individuals impacted by the change.

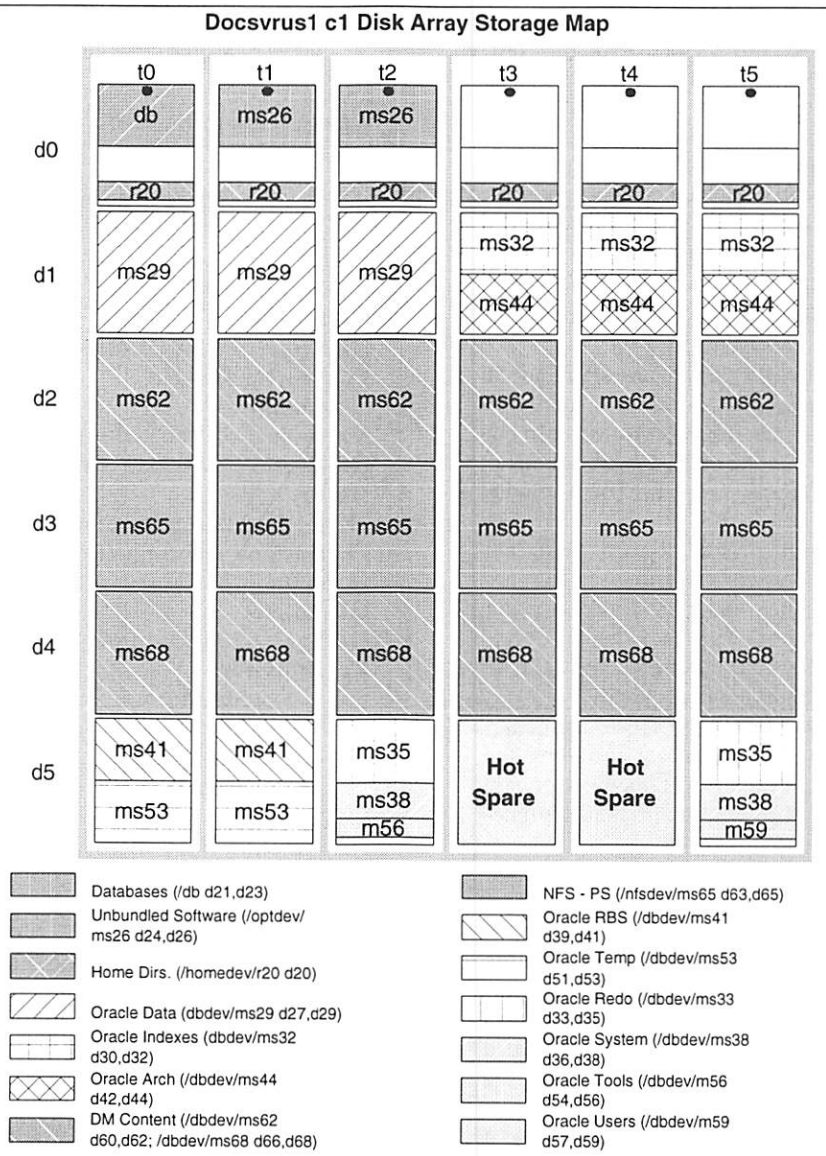

Figure 2: Disk Layout Example.

8. Revisiting the defined roles and responsibilities of system personnel.

Establishing & Documenting the System's Base-Line

Before a change is made to a production environment the current state of that environment must be well understood and documented. If the current state is not documented, then all work on the change implementation must halt and the current system state must be documented. It is important to know the state of the computer system in order to properly evaluate the effects of the change. It is just as important to have a defined state to which the system can be restored should it become necessary to back out the change. The documentation specifying a system's base-line should include at least the following:

- **Server Configuration**. This includes the server component layout, disk layout, platform type, amount of memory, number of CPUs, etc. The example in Figure 1 shows a server component layout, platform type, amount of memory, and number of CPUs. An example disk layout is provided in Figure 2 for one of the disk arrays in Figure 2.
- **Software List**. This list includes all major software which has been installed on the system including versions and patch levels. The example in Figure 1 lists the major software packages installed on the system.
- **Roles and Responsibilities**. Each role required to support the computer system is included in this document and its responsibilities defined. The roles should include staff, system administrator(s), dba(s), client(s), application administrator(s), etc.
- **Network Diagram**. The network diagram shows how the system's various nodes are related to each other and the communication path(s) between them. This includes all development, pre-production and production clients and servers, as well as support systems.
- **Installation Steps and Logs**. This documentation includes the steps taken for installing the current operating system and major software applications, and the logs of the actual procedures.
- **Standard Operating Procedures**. A SOP defines the procedure for accomplishing a specified, encapsulated task. It includes all relevant information and instructions detailed enough for an entry level system administrator to perform the procedure. A SOP is written for all common changes to and procedures performed for the computer system. An example of the former is replacing a faulty disk. An example of the latter is performing routine monitoring of the system. Figure 3 shows A SOP template.
- **Change Control Form**. A change control form (CCF) is created to document any change not covered by a SOP if that change has a reasonable chance of impacting use of the computer system. Figure 4 presents a CCF template.

Procedure Name
Standard Operating Procedure
Author
Date

I Procedure Description
II Scope of This SOP
III Complexity
IV Estimated Outage Time
V Backout Contingency Plan
VI Note(s)
VII Supporting Documentation
VIII Procedure
Step 1
Step 2
. . .
Step N
IX Test Plan

Figure 3: Standard Operating Procedure Template.

Change Name
Change Control Form
Author
Date

I Problem Definition
II Solution
III Scope of This Change
IV Project Team
V Complexity
VI Impact of Not Making Change
VII Estimated Outage Time
VIII Priority
IX Re-Validation Recommendation
X Backout Contingency Plan
XI Status
XII Note(s)
XIII Supporting Documentation
XIV Procedure
Step 1
Step 2
. . .
Step N
XV Test Plan
XVI Notes on Implementation

Engineer: _____ Date: _____
Customer Rep: _____ Date: _____
QC Specialist: _____ Date: _____

Figure 4: Change Control Form Template.

- **Backup Procedure**. This document includes a backup schedule indicating the backup levels, a retention schedule, and off-site storage arrangements.
- **Disaster Recovery Procedure**. The list of potential disasters ranges from losing or corrupting data and failure of a system component

to losing an entire site. While not every conceivable disaster can be reasonably accounted for, there should be (3-5) procedures that cover various, reasonable points within that range. Given such a set, it should be possible to find a procedure which addresses a disaster similar in nature to an actual disaster.

- **Test Scripts**. The application Quality Assurance/Quality Control [4] (QA/QC) group should provide an appropriate set of test scripts to be run after any change which impacts the application. As Figure 5 shows, a *test script* is a list of tests to be performed and for each test the expected results, a place to indicate the outcome of the test, and a comment section.

Test Script Name
Test Script
Author
Date

Test List
Test 1
Description
Expected Results
Actual = Expected? (Responsible Initials):
Yes: No:
Comments
.
.
.
Test N
Description
Expected Results
Actual = Expected? (Responsible Initials):
Yes: No:
Comments

Figure 5: Test Script Template.

Additional items that may be documented are discussed in [5]. Once a system's base-line has been established and documented, attention can be turned to understanding the change to be made.

Understanding the Change

Before a change is implemented on the production system, it is necessary to understand the change. The better a change is understood, the better it can be implemented and maintained. Many questions need to be answered. What problem is being solved? How does the change solve it? What are the side effects of making this change? What are the implications of not making this change? Who is affected? How are they affected? What is the reliability of this change? What training will be needed as a result of implementing the change? What is the best way to make the change? What is the scope of the change? Who is experienced in this area, with this change? That is, who can help if an unforeseen problem occurs during or after the change? All changes should be well thought out beforehand. This process can be greatly facilitated by having a change control form (see Figure 4) which brings these questions/issues, as well as other's, into consideration. Once a change has been *thought out* it is tested on a non-production system.

Testing the Change

Testing the Change on a Test System

In order to fully understand a change, including how best to implement it and the effect it will have, it is necessary to *make* the change. First, all appropriate documentation is read, including any vendor provided information. Then, an initial Work Plan (WP) is developed. Finally, the change is implemented on a non-production system by following the initial WP, an example of which is given in Figure 6.

At this phase a separate test system is preferred over a common test system or a development system in order to minimize the change's impact on others. However, a common test or development system may be used as long as the impact on users of this system is taken into consideration. This initial testing of the change is not performed on a production system as the change's impact may not be fully understood at this time.

Testing the Change on a Pre-Production System

Once the change itself is well understood and the work plan has been updated from the previous stage, the change needs to be tested within the production environment. However, the test at this stage is performed on a separate pre-production system. This system should exactly match the production system in every way possible, certainly in every way that matters. The development or test system can not be used for this purpose because there are usually numerous differences between these systems and the production system. Developers and system administrators need an environment where they have the freedom to learn and implement new systems, subsystems, ideas, etc. On the other hand, production users need an environment that is reliable, available and supportable [6]. These are very different environments and should be kept separate.

Since the development or test system is significantly different than the production system, this final testing of the change is completed on a pre-production system. This testing includes executing the appropriate QC generated test scripts. These test scripts should provide evidence that the system resulting after implementation of the change still meets the specified QC requirements. Once all testing is complete, the final work plan is created.

Documenting the Change

To keep the current state of the production system documented we must document all changes. Control of a system is kept only to the extent of detail with which each change is documented [7]. This is best done with a combination of a change control form, SOPs, work plans and change logs. Before the change

is implemented on the pre-production system the change control form is filled out and the initial work plan is created. The change control form and work plan differ in that the change control form is written for the client and reviewers and contains a high level description of the change implementation. The work plan is written for the system administrator(s) who will be executing the implementation and is a much more detailed description of the procedure. Before the change is implemented on the production system the final work plan is created. As the change is implemented on the production system the actual steps taken, which may differ from the final work plan for unforeseen reasons, need to be logged. Afterwards all other impacted documentation (for example, SOPs) needs to be checked and updated as necessary.

Reviewing the Change

It is always a good idea to have a change reviewed by peers, manager and client via the change control form, work plan and a presentation. Peers are usually the best at understanding, and hence critiquing, the computer system's technical aspects of the change and the overall approach. Furthermore, the better that peers understand the change, the better they will be able to assist in making the change. A manager brings to the table a global view concerning the impact of the change. Furthermore, it is important to have a manager's support of the process. A client representative knows well how the change will impact the client's specific processes and environment, as well as the relative priority with which the client views the change. The more involved a client is in the change process, the more confident they and the system

Updating DocMgmt Solaris 2.5.1 Patches
Work Plan
3-September-1997

Project: Updating DocMgmt servers Solaris 2.5.1 patches.

Scope: The patches will fix known operating system problems.

Purpose: To make DocMgmt production servers more reliable.

System Engineers: Frank Northrup (DCCI)

Estimated Outage Period: 4 hours.

Plan:

Preparation

1. Schedule outage to update patches.
2. Compare current patches to new patches to be certain all are covered as needed. [<server>: ls /var/sadm/patch compared to docsvrus1:/opt/adm/Sol251/Patches/]
3. Create list of patches to uninstall. Create backout_patchinfo and backout_patchlist files for "backoutpatches" script. Reverse the order relationship used to install the patches
4. Create list of patches to install. Create patchinfo and patchlist files for "installpatches" script. Keep the order relationship used on other servers.
5. Verify that backups are good the morning before starting the patches update.

Updating the Patches

1. Stop the Documentum and Oracle processes.
   /etc/init.d/dbdm stop
   /etc/init.d/dbdmbroker stop
   /etc/init.d/dbora stop
2. Move automatic Documentum and Oracle start/stop links to Hold directories.
3. Mount the administration filesystem from Docsvrus1 and go to appropriate bin directory:
   mount docsvrus1:/opt/adm/Sol251 /mnt; cd /mnt/<server>/bin
4. Backout the patches:
   ./backoutpatches < backout_patchinfo-full-path-name>
5. Reboot system:
   shutdown -y -i6 -g0 "Rebooting to remove old patches."
6. **Checkpoint:** If reboot fails in any way, diagnose and resolve, or return to base-line state.
7. Mount the administration filesystem from docsvrus1:
   mount docsvrus1:/opt/adm/Sol251 /mnt
8. Install new patches:
   cd /mnt/<server>/bin; ./installpatches <patchinfo-full-path-name>

⋮

14. **Checkpoint:** Make sure Oracle and Documentum systems started properly. Make certain client application runs fine. If not, diagnose and fix, or return to base-line state.

Figure 6: Example of a Work Plan.

administrator will be in the reliability and consistency of the resulting system. This will, in turn, foster a good working relationship between system administrator and client.

Defining a Back-Out Strategy

Although following the change control processes mentioned above greatly reduces the chance of the change causing problems on the production system, something may still go wrong. Thus a back-out strategy needs to be defined prior to implementing the change on the pre-production system. This back-out strategy should take the system back to its base-line state. The back-out strategy itself should be well-documented as part of the change control form and work plan.

Training All Individuals Impacted by the Change

All folks affected or potentially affected by a change need to have training on what the change is and how it impacts them. If a change results in a corresponding change in how clients use the system, then they will need training so that the time it takes to use the system again at the same level or better (if the change resulted in an enhancement to the system) is minimized. Support personnel (staff, system administrators, operators, etc.) need training so that they can continue a high level of maintaining the system and supporting users of the system.

Change Name
Change Management Checklist
Author
Date

Date	Item	Comments
_____	☐ Base-line established	_____
	☐ Server configuration	_____
	☐ Software list	_____
	☐ Roles and responsibilities	_____
	☐ Network diagram	_____
	☐ Installation steps and logs	_____
	☐ Standard operating procedures	_____
	☐ Change control form	_____
	☐ Backup procedure	_____
	☐ Disaster recovery procedure	_____
	☐ Test scripts	_____
_____	☐ Change control form	_____
_____	☐ Work plan - Initial	_____
_____	☐ Test on test system	_____
_____	☐ Review - Client	_____
_____	☐ Review - Manager	_____
_____	☐ Review - Peers	_____
_____	☐ Back-out strategy	_____
_____	☐ Test on pre-production system	_____
_____	☐ Work plan - Final	_____
_____	☐ Training - affected individuals	_____
_____	☐ Roles and responsibilities revisited	_____
_____	☐ Implemented on production system	_____

Figure 7: Example of a Change Management Checklist.

Revisiting Roles and Responsibilities

Finally, the roles and responsibilities which were defined with respect to the production computing system must be re-evaluated. Some roles may have been obsoleted by the change and/or new responsibilities or roles may need to be filled. In any case, all those affected by the change need to be involved in the evaluation and redefinition of the roles and responsibilities.

Discussion

There are several elements to the approach presented in this paper. In order to help keep track of the completion of these elements a Change Management Checklist is provided in Figure 7. This approach is being used successfully by DCCI with its clients. At one site there have been only five unscheduled outages in 2.5 years for 17 production, pre-production and training UNIX servers with 44 CPUs, 9.5 GBytes of RAM and 1.7 TBytes of disk space. This set includes: one production server with 8 CPUs, 1.5 GBytes of RAM and 0.3 TBytes of disk space; a second with 8 CPUs, 1.0 GBytes of RAM and 0.5 TBytes of disk space; and a third with 6 CPUs, 1.5 GBytes of RAM and 0.5 TBytes of disk space.

At first, it seemed that this approach would improve reliability and availability of production systems, but with the trade-off of significantly increasing the amount of time and effort spent by the system administrator on any given change. However, what we have found is that while the initial effort is greater, we actually save time and effort:

- when repeating a change a month or more after a previous occurrence;
- when transferring knowledge to other system administrators performing the same or similar change;
- when filling in for the primary system administrator;
- by assisting in promoting site consistency;
- by assisting in promoting site standards;
- when diagnosing problems (because everything is already documented).

Conclusion

Bug-free software and transparent changes are a system administrator's ideal scenario. Unfortunately, these are rare occurrences. Further complicating matters is an environment in which employers are reducing their computing system support personnel while increasing the number of systems which need to be supported. It becomes increasingly important that system administrators take and keep control of the systems for which they are responsible. The extent to which a system administrator is able to control and manage change is the extent to which they will be able to provide a reliable, available and supportable computing system to their clients *and* maintain their sanity. This paper describes a platform independent approach for doing just that.

Acknowledgements

Special thanks to Kevin Suboski, owner and President of DCCI, for leading us down this path. Thanks to Kevin, fellow DCCI employees and Pharmacia & Upjohn, Inc. employees for encouraging us during the writing of this paper. Many thanks to Pharmacia & Upjohn, Inc. for providing an environment in which this approach could be developed, tested and refined. Thanks also to our LISA '97 reviewers for their insightful comments and suggestions. Thanks to both of our families for their loving support.

Author Information

Sally J. Howden is currently an Open Systems Consultant with DCCI. She has worked as a UNIX system administrator or consultant for six years. She earned a B.S. in Computer Science and Mathematics from Calvin College, a M.S. and a Ph.D. in Computer Science from Michigan State University. Sally can be reached via email at sallyj@distcom.com .

Frank B. Northrup is currently an Open Systems Consultant with DCCI. Previously he managed the computing facilities for the CS and CIS departments at Michigan State University and Ohio State University, respectively. He has managed UNIX systems for ten years. He earned a B.S. in Computer and Information Science from Ohio State University. Frank can be reached via email at frank@distcom.com .

References

[1] Armour, J. and W. S. Humphrey, "Software Product Liability," Software Engineering Institute, TR CMU/SEI-93-TR-13, ESC-TR-93-190, August, 1993.

[2] "Quality Systems: Part 13. Guide to the application of BS (British Standard) 5750: Part 1 to the development, supply and maintenance of software," BS 5750: Part 13: 1991, ISO 9000-3: 1991, BSi Standards.

[3] "Computer Validation Policy for Pharmacia & Upjohn," Pharmacia & Upjohn, Inc., Version 1.0, April, 1996.

[4] "IEEE Standard for Software Quality Assurance Plans," IEEE Std. 730-1984.

[5] Nemeth, E., G. Snyder, S. Seebass and T. Hein, *UNIX System Administration Handbook,* Second Ed., Prentice Hall PTR, Upper Saddle River, NJ, ISBN 0-13-151051-7, 1995, pp 10, 741-742.

[6] Kern, H. and R. Johnson, *Rightsizing the New Enterprise – The Proof, Not the Hype,* Sun Microsystems, Inc., Mountain View, CA, U.S.A., ISBN 0-13-132184-6, 1994.

[7] Agalloco, J., "Computer System Validation – Staying Current: Change Control," *Pharmaceutical Technology*, January, 1990.

Developing Interim Systems

Jennifer Caetta – Jet Propulsion Laboratory

ABSTRACT

One of the recent challenges in the aerospace industry has been to smoothly transition operations-oriented computer systems to meet increasing demands on smaller budgets. Sometimes the best solution is not affordable, but the current situation is equally untenable. Change becomes necessary, but is only acceptable at minimal operational impact and financial costs.

Frequently, interim software solutions must be developed while new hardware is under design or production. These interim systems are characterized by an immediate need with a limited budget. This paper discusses several factors involved with producing a working software set in minimal time, such as the initial approach, budgetary concerns, and resource-scavenging. By recognizing the importance of these factors, interim system developers will be able to produce working systems with a minimum of setbacks.

Introduction

Software should be perfect. Networks should be capable of handling multiples of the maximum expected traffic levels. Hardware should never fail. But "should" will not release a system before an operational deadline, and it invariably costs more than your budget can afford. The solution is to instead create a system which meets immediate customer demands, yet leaves an open path for future modifications.

The ground data systems Jet Propulsion Laboratory range from state-of-the-art to nearly obsolete. These older systems have no funding for hardware upgrades (replacement hardware is expected within five years), yet they still support spacecraft on a daily basis. This paper focuses on methods by which the operational capabilities for these systems can be significantly extended at a minimum cost.

These methods consist of three main stages: initial approach and design, working within and around budgetary limitations, and resource-scavenging. While these methods may seem obvious, the impact of not following these steps is usually only understood in retrospect. Therefore, this paper will set forth guidelines in each of these areas and examine their usefulness with three operational interim systems (currently in use at JPL for their Radio Science Systems Group).

Program Descriptions

Getting Real-Time Data: get_tss

The first of the three interim systems which will be used in later examples is a method of getting real-time data from the central communications facility at JPL (the GCF) through a firewall and into a display package which ran on a Sun 4/330. The original system used a dedicated modem line to a Prime 4050, which filtered the data based on spacecraft and receiving station. The data was then sent via TCP/IP to a Sun 4/330 and processed into shared memory. Users could then display various parts of the data. In order to change the filter to use a different spacecraft or receiving station, the program had to be stopped and restarted with different parameters.

When the interface was to be formalized between the Radio Science Group and the GCF, the possibility of using Ethernet was considered politically unacceptable. It was decided that X.25 could be used to replace the modem interface, because it was already in use in several places on-lab. A filtering program was written for the incoming data which would allow filter parameters to be changed on-the-fly, and the routines to handle data receipt were isolated from the rest of the software – to facilitate future changes in the incoming data protocol. This system worked reasonably well; however, differences in Sun X.25 and Encore X.25 (the GCF hardware) soon demonstrated a problem: when our system was not listening to the Encore, the packets were dropped "on the floor" in the GCF, much to their dismay.[1]

It was soon suggested that management reconsider the possibility of using a UDP/IP connection, outbound data only. Upon their eventual approval, the adaptation was simple. Routes were created to allow only one-way traffic on a single unix socket, and the X.25 receiving software was modified only to the extent of creating UDP/IP sockets instead of X.25 circuits. Figure 1 illustrates the three data paths used during the transition.

The work described was performed at the Jet Propulsion Laboratory, California Institute of Technology under contract with the National Aeronautics and Space Administration.

[1]Note: this is not a problem in Sun-to-Sun X.25 connections.

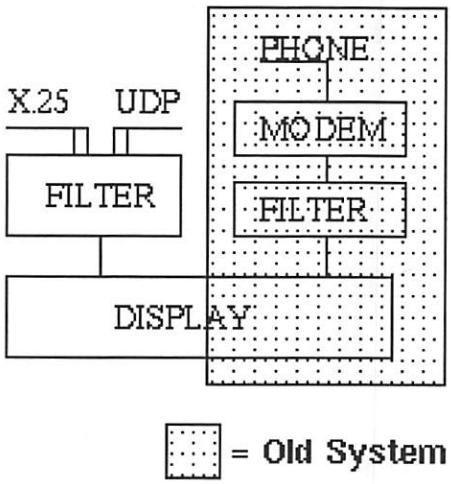

Figure 1: get_tss, original system and current system.

Remote Control of DSN Hardware: remops

The second of the three interim systems remotely controls the open-loop receiver (a digital signal processor with several physical components, commonly called the DSP) at the Deep Space Network (DSN). It allows users at JPL or Stanford to configure the DSP as well as to set up unattended operations-based configuration and run scripts. The DSP includes MODCOMP 2000 and several dedicated boards, running REAL/IX (a real-time Unix-based OS). The replacement hardware for this receiver will be fully funded for the Cassini project, but not until 2002. The unattended operations capability was initially implemented in order to support a grueling 3-year, 8,000-occultation mapping phase by Mars Global Surveyor.

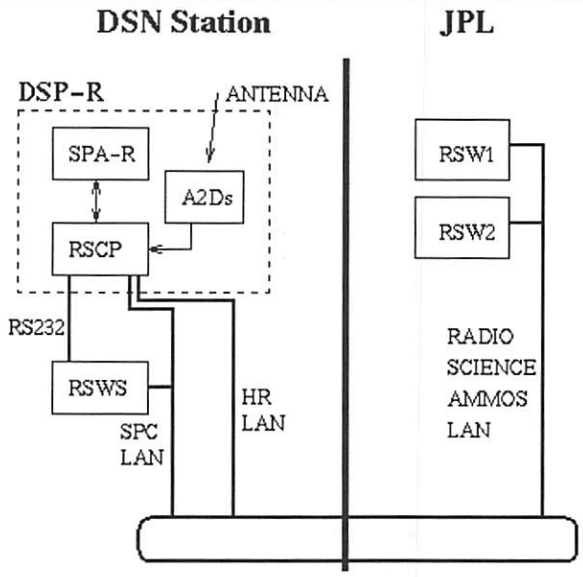

Figure 2: remops Hardware Layout

The operational "remops" system uses a Sun Sparc 5 connected via an RS-232 serial cable to the MODCOMP, using the diagnostic terminal connection as a "back door" into the system. A fairly standard client/server package handles interactions with remote users. Figures 2 and 3 show this layout.

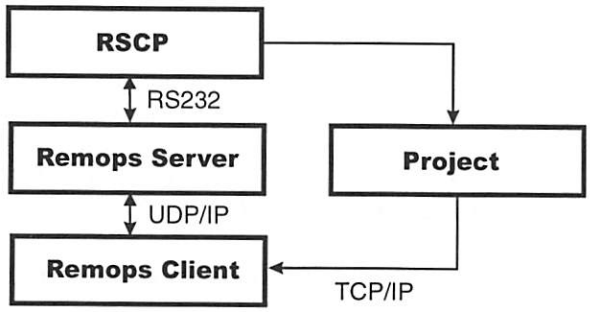

Figure 3: remops Software Data Path

Data Analysis: RSVP

The last of the three systems is a data analysis tool which originally was meant to replace the functionality of a large number of Fortran programs which ran on a Prime 4050 with an array processor. Many of these programs were simply copies of another program with minor modifications or refinements – spaghetti code with names like "resid01", "resid02", and "resid03" to distinguish the versions.

The overall structure of the Radio Science Validation and Processing tool, (rsvp), is a C++ "shadowbox."[2] Figure 4 shows the overview of the design. The front end, which must read in many different data formats extracts the required data and makes it available to any of the successive programs. The types of processing were categorized, and their arguments incorporated into the GUI (graphical user interface), but the programs are called with system[3] calls. This not only allows the scientists to continue modifying their programs, but it allows the program to remain stable in overall structure. New data formats only require another C++ class and a few header file entries; the rest of the code remains fixed. New programs require only a new system call and a GUI-incorporated argument list. Version numbers are kept on the scientist-provided subroutines, and only one version is "supported" at a time. However, multiple entry points into the program are available to accommodate the occasional need to run non-supported programs on the data, then re-insert that data back into the program.

The Initial Approach: How Does Your Project Grow?

The most crucial stage of any new project development is the initial communication of ideas and goals, and the understanding of how the project will

[2]Shadowbox: a decorative set of small attached boxes which is hung on a wall. Knick-knacks are then placed on each little shelf.

[3]Standard C library system command, SunOS 4.1.4.

grow and evolve. Several papers and books have been written on strategies to formalize and elaborate each aspect of clearly communicating requirements and limitations [1, 5]. Software process models and win-win [2] negotiation methods are continuing to expand by accommodating "real world" behaviors [4] and becoming more generalized.[4] However, some projects – interim systems in particular – are very rarely ever "done." These systems are created to fill a basic need, and are usually expected to be replaced or even retired. But what if an interim system has years before that happens? The possibility of added capabilities and refinement at a minimal cost can be very tempting.

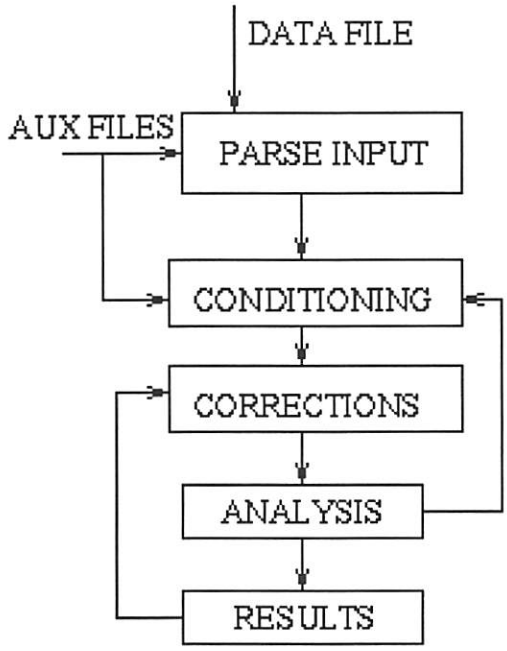

Figure 4: rsvp Software Data Path

Interim systems therefore tend to evolve into a series of stacked spiral process models [3], something which could be described as a "tornado model." Each layer inherits a base set of risks and dependencies from the lower layers, but is uniquely linked to a specific capability which is to be implemented. Capability requestors can be users, management, or "evolutionaries" – developers who wish to modify code now to make future instantiations easier to develop. The capabilities themselves usually do not interfere with the functionality of previously-developed instantiations, although such conflicts are usually identifiable through the inherited dependencies.

Because of this penchant for growth within the low-budget, fast-turnaround and high-risk[5] framework which characterizes interim systems, several of the normal developmental stages – communication, realization of risk, and designing for reusability – become high-risk factors themselves. Without an awareness of the evolutionary factors within each stage, a design could result in a program too inflexible to accommodate future changes which might have been anticipated.

Communication

The clear communication of objectives between developers and requestors is perhaps the most significant method of preventing a failure to successfully implement a software package. When developing a system which is expected to encounter new-capability requests, the communications and negotiations methods mentioned above become even more critical. By "getting close to the customers" [7], and actively searching for possible future requests, the design of a system can anticipate the need for certain types of flexibility with a much larger lead time than would otherwise be available. Weekly meetings to discuss status, problems, and possibilities can not only achieve Gilb's "risk sharing principle" [6], but makes sure that no misunderstandings have occurred and that both developers and requestors understand the overall direction of progress. In addition, the future use of the system can be addressed; the design can then inherently facilitate the eventual implementation of new capabilities.

As an example, the rsvp program, described above, was not simply a software port to a different architecture; it was a complete re-design in structure and responsibility. The package needed to smoothly accept multiple input data formats, ensure easy the implementation of new input format handling, and allow the scientists who originally developed the analysis programs to continue to develop them to accommodate new frequency band equations, new antenna combination methods, or new analytical ideas.

The "shadowbox" design of rsvp, described above, satisfied the above conditions. The design also made it easier to incorporate the most important portions of the Prime 4050 original software, thereby creating a basic-use capability quickly. However, one important design stage did remain unrecognized until well after the main structure had been completed: the value of continued discussions with the users about the difficulties they were having or what kind of functionality they would like in the future. As it turned out, some of what they were asking for was fairly simple to accommodate, yet they had written separate programs to bypass that need.

By modeling the package with the consideration of how the software would grow and develop (using a "tornado"-style model, Figure 5), this extra effort

[4]However, actually applying these models to a proposed system always requires some tailoring – even if that tailoring consists of recognizing each phase but not acting upon it.

[5]In this case, the risk associated with the uncertainty involving legacy systems.

could have been avoided. However, after adapting the software upgrade procedure to reflect this type of model (incorporating weekly meetings, feedback sessions, and future-use discussions), we found that developers were proactively tailoring the code to facilitate the eventual implementation of these "future-use" capabilities in parallel with implementing changes requested through the feedback sessions. This synchronous development did not greatly increase the time needed to complete the feedback requests, and when the decisions were made to implement some of these capabilities many months later, the time involved was much less than anticipated because of this software "priming."

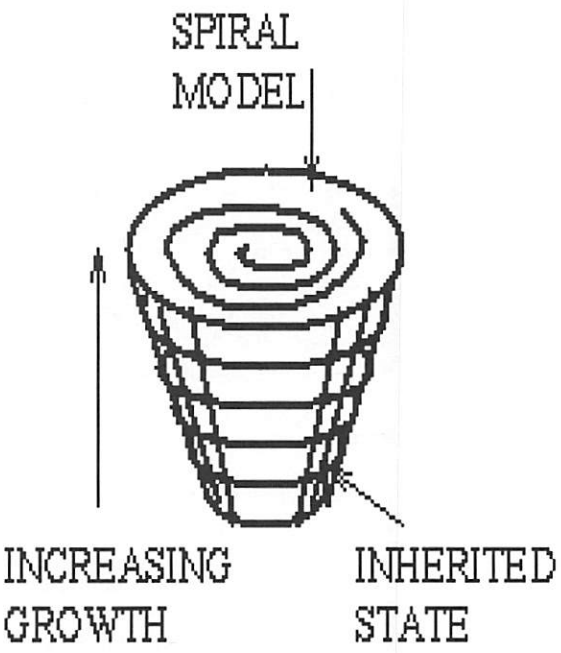

Figure 5: "Tornado" Model

Realization of Risk: Just Don't Panic

Interim systems have an inherent high-risk factor: the high probability that the system will need to interface or interoperate with a legacy system. These systems have a penchant for undocumented behavioral anomalies, or "features." Even the best risk assessments underestimate this potential.

For example, while testing the terminal interface between the DSP and the remops program, two such "features" materialized. The first was the assumption in the terminal session software that no human would sit down and type 1024 commands into the system (the terminal session was not the primary command interface, but a diagnostic back-door). However, the human would (obviously) want a printout of all the commands entered. Therefor, all commands were stored in a buffer – a buffer with 1024 spaces. Upon reaching the 1024th command, the system refused to take further input until the terminal session was aborted.

This limit led to the idea to open up a new terminal session whenever a new configuration set needed to be input (which uses approximately 15 commands). Within five minutes of back-to-back test runs, we discovered a memory leak in the DSP system queues; the more terminal sessions started and stopped, the less memory was available for the rest of the system. Eventually the entire system would hang. Therefore the only compromise was to keep track of how many commands got sent, and only when that number neared 1024, to automatically shut down and restart the terminal connection. This would give the system the maximum possible length of time in which to operate before the system would need to be rebooted: about five weeks.

Luckily, the hardware rarely was able to operate for three weeks without needing to be rebooted for other, less understood, reasons.

While it is very difficult to accurately model this type of high-risk behavior, interim systems designers do have some leverage when proposing solutions which would otherwise not be considered. Support from management must be strongly committed, especially when the proposed solutions are best described as "creative." Don't panic: interim systems projects are rarely undertaken when they are not urgently needed. While not exactly intended in the same context, Gilb's uncertainty motivation principle [6] can be applied to the potential for a lack of support from management:

> Uncertainty in a technical project is half technical and half motivational, but with good enough motivation, uncertainty will not be allowed to lead to problems.

Frequently, the thought of an interim system not working provides plenty of motivation.

Designing for Reusability

In general, the expectation upon an interim system is that it will be in use only until the replacement system arrives. However, this expectation cannot defend throwaway code, nor code which is not portable or upgradeable. The replacement system may not be available until much later than expected.

In anticipation of the change, however, designers should understand the expected replacement system, not necessarily as a final destination in the growth of the interim system, but more as a system which is trying to surpass the possibilities of the interim system. Time should not have to be spent redeveloping the modules which create old capabilities! Instead, by writing code which is easily reused and portable, modules of the interim system can be incorporated into the replacement system, allowing more time to be spent on those previously unreachable capabilities.

In this way, the interim system becomes a resource which can be scavenged later, with a minimum amount of portability and modification effort. The get_tss program, described above, exemplifies this through the X.25 to UDP/IP transition; the body of the code was unchanged while only the socket initiation code needed to be changed.

Budget Limitation: Creativity While Being Broke

The second guideline towards successful interim system development is creative financing. Usually, the most insidious of budget allocation patterns is the tendency to think of financial sources in set, well-used ways. At JPL, spacecraft projects tend to think in terms of "project money": what that money buys, the project owns completely. In companies with multiple projects, this can lead to a large amount of replication of resources with minimal gains in reliability.

At JPL, the role of Radio Science is (given a spacecraft, a signal, and a receiving station) to produce planetary/atmospheric science data. As projects come and go, the Radio Science Group needs to continually expand and grow in order to accommodate these changing needs and be a truly multi-mission service provider. Because of this growth pattern, the Radioscience Operations and Data Analysis Network (RODAN) can can itself be characterized as an interim system. However, mainstream, project-line funding can be severely detrimental to the overall quality (robustness, reliability, capabilities, and veracity) of the Radio Science system.

As an example, until last year the low number of concurrent projects had allowed RODAN to follow the same trend as the projects: each project owned a subnet off of the central operations backbone, and all workstations associated with that project were installed on that subnet (See Figure 6). Because Radio Science was not located in the same building as the project offices, however, fiber lines had been installed between buildings: one for Galileo, and one for Mars Global Surveyor (MGS). Each project was willing to buy Radio Science a single workstation for real-time operations and sequencing activities. This was all in accordance with the budget limitations of each project, and all very separate from each other.

When the operations and sequencing scenarios for Cassini were initially considered, this trend showed to be extremely restrictive. The expected future increase in concurrent mission numbers (e.g., Pluto Express, Mars 98) and the possibility of a single-point failure associated with the one workstation/one project funding scheme emphasized further the need for a major design change. Politically, however, no funding was available: Cassini was thinking of duplicating the entire Radio Science team within the Cassini hierarchy, MGS had already spent all it could on Radio Science by funding the remops development, and Galileo was settled into the status quo.

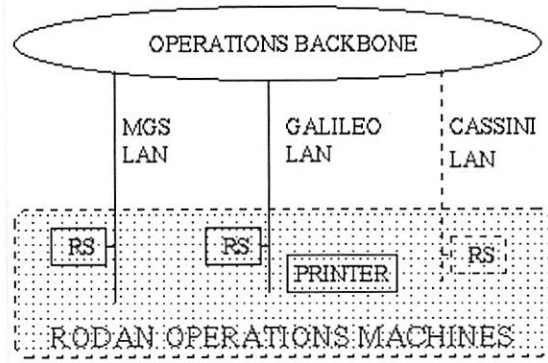

Figure 6: Original Radio Science Operations Network Design

The basic characteristics of interim systems, however, provide a unique source of justification of financial expense. Invariably, interim systems are strongly needed – and not always only by the proposing group! By combining a financial analogy of Gilb's uncertainty motivation principle with a proof of lowered future costs, along with a "sale-by-example" approach, both the Cassini and Galileo projects realized that they, too, needed the remops system. However, both projects were also made very aware of the vulnerability of the current single-point failure design, so when the final layout of a single Radio Science LAN (Figure 7), in parallel with the individual project LANs, they were more willing to trade a project-owned system for an affordable and robust shared system. MGS would give their fiber line back to JPL, Galileo would donate their fiber and hub for general Radio Science use, and Cassini would fund labor costs. The benefit for modifying the strict project-line funding would come in operations cost: with the remops system, 30 minutes of pre-pass calibration are no longer needed. At the 70-m antennas, time costs about $1200/hour, and at the 34-m stations, around $750/hr. Obviously, the savings build quickly.

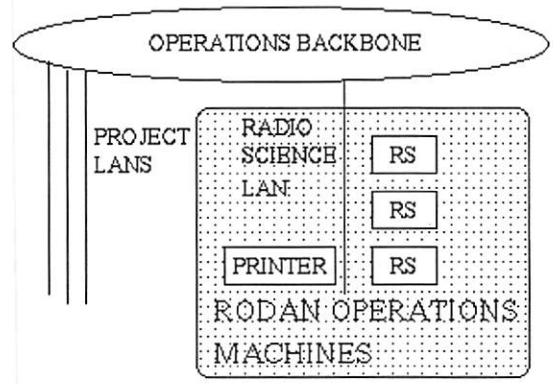

Figure 7: New Radio Science Operations Network Design

Admittedly, the feat of pulling financing out of apparently thin air does not always work out as

cleanly as that example. While the diplomacy, approach, and presentation of a different type of financing remains an art form, interim systems, however, give developers a lot of leverage. For example, when dealing with the burgeoning Cassini "fiefdom" design, the Radio Science group can really only emphasize its "proven and stable" contracting firm image, and hope for the best. However, if Cassini wants the remops system to be ported to HP/UX, or utilize the operational experience which resides within the Radio Science group, then financial support would obviously be expected.

Resource Scavenging: Scotty[6] had the Right Idea

The third guideline crucial to the development of interim systems is resource scavenging – essentially a protection against wasted time. With a little time invested in identifying available resources, a lot of time can potentially be conserved. Most of this scavenging phase is already a part of the spiral software model – such as finding appropriate third-party applications. However, interim/legacy system interfaces have a high cost risk associated with not emphasizing certain types of resources – freeware/COTS applications, original developers, and creative perspectives.

Tools

The primary source of available tools is from other programmers or developers – and for interim systems, the fact that stable freeware is much more affordable than many COTS packages is not lost in the design phase. While these packages may not exactly fit the system's needs, adaptation or extension can frequently be added at a fraction of the cost of writing a comparable system.

For example, while writing a graph tool for the analysis package rsvp one of the developers was able to get a lot of ideas and examples from existing software, such as the Itcl source[7] and documentation, and the associated extended widget libraries. This saved him a lot of time; he was able to produce a working product within a few weeks.

People

Original developers of the legacy systems with which the interim system must interface are also extremely valuable. While many anomalies may not be documented, these developers may be able to give information about the problems and advice on how to work around them. Unfortunately, these developers are frequently gone or unavailable for extended periods of time, or in one case "wouldn't touch that system with a 10 foot pole, except to hit it very, very hard."

A lack of these human resources can severely impact the effectiveness of an interim system because of the close ties between legacy system interfaces and undocumented behavior. For example, while developing the remops software, the interface to the DSP proved to be troublesome because of some assumptions about how the communication between the MODCOMP and the secondary system (the Signal Processing Assembly – Radio Science, or the SPA-R) was conducted in the case of a reboot of the SPA-R but not of the MODCOMP. It had been understood that the MODCOMP would re-send the configuration settings to the SPA-R, but this was not exactly the case; the MODCOMP only sent over a subset of these configurations. This was not discovered until the system was in operational use: such a reboot pattern does not happen frequently, and it had not been tested because it technically wasn't a part of the remops system.

In these cases the next best resource is a user-group maintained log of "unexpected" observed behaviors. The documentation of interim systems frequently must cover problems with the legacy systems; with such a log, the same problem won't bite you (or another developer) twice.

Creative Perspectives: Does This Really Have To Be This Way?

The last segment of resource scavenging is to identify which applicable technical restrictions are easier to change than avoid, and which are easier to avoid than to change. Interim systems are frequently subjected to technical restrictions which are based on older systems, and which may not be not completely relevant. Therefore, the design of the system must take into account the probability of a change in these restrictions, and factor in the benefits of using the resources which would then become available.

While originally implementing the get_tss program, Cisco routers were not trusted to provide a firewall when outbound UDP/IP data was allowed; therefore the X.25 interface was selected. The design of the package, however, ensured that if an ethernet connection was eventually allowed, minimal changes would have to be made. Over the course of a year while the communications facility's staff became more familiar with the Cisco routers, and given the problems encountered with the Sun/Encore X.25 interface, the system was able to be switched to UDP/IP, and the extra cables, cards, and switches were able to be discarded. In this case, the restriction could not be avoided, and it was therefore easier to simply change the software when that restriction was eventually lifted.

On the other hand, the remops system is not able to get monitor data via broadcast at the remote workstations at Stanford due to valid security restrictions. In this case, the restriction is avoided by designing several parallel methods of receiving monitor data, and using the best available method for each machine:

[6]The engineer from Star Trek (the original series).
[7]Itcl2.0 for these packages.

via broadcast, database query, or by a special query to the DSP itself.

Conclusions

While the spiral software model has incorporated the vast majority of system development and design concerns, interim systems require extra emphasis on the initial approach, budget considerations, and resource scavenging. Because of the nature of such systems – interfacing with legacy software and fragile hardware, intended for short-term use and therefore needed on an per-capability basis – ignoring the importance of these factors can lead to serious consequences, such as late-stage design changes, unexpected incompatibilities, or worse, not having a working product if the old system fails before the new system has been funded. Essentially, interim system development theory can be summed up by the simple phrase: get it done, make it work, and don't spend any extra money. By following the above model, developers can build working, growing interim systems with just that characteristic.

Author Information

Jennifer Caetta graduated from the University of California at Berkeley in 1993 with a Bachelor's Degree in Electrical Engineering/Computer Science. She joined the Jet Propulsion Laboratory, where she is an applications developer, systems administrator, and configuration manager. Reach her via US mail at Jet Propulsion Laboratory, M/S 264-325, 4800 Oak Grove Dr., Pasadena, CA 91109. Reach her electronically at caetta@rodan.jpl.nasa.gov.

Bibliography

[1] B. Boehm, "A Spiral Model of Software Development and Enhancement", *Computer*, May, 1988, p. 61-72.

[2] B. Boehm, "Theory-W Software Project Management: Principles and Examples", *IEEE Transactions on Software Engineering*, 1989.

[3] B. Boehm and F. C. Belz, "Applying Process Programming to the Spiral Model", *Proceedings of the 4th International Software Process Workshop*, May, 1988.

[4] B. Boehm and P. Bose, *A Collaborative Spiral Software Process Model Based on Theory W*, Aug., 1994.

[5] R. Fisher and W. Ury, *Getting To Yes*, Houghton Mifflin Company, 1981.

[6] T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley Publishing Company, 1988.

[7] T. J. Peters and R. H. Waterman, *In Search of Excellence*, Harper and Row, 1982.

A Large Scale Data Warehouse Application Case Study

Dan Pollack – America Online Inc.

ABSTRACT

Large data warehouse applications are beginning to become more necessary as large amounts of data are collected from the day to day interactions of businesses and their customers. Businesses are now using these enormous amounts of data to decide how and where to advertise, what resources projects will require in the future, and to gauge the general direction of the company to keep it on track.

This paper is a case study of the design and implementation of a 1+ terabyte data warehouse for marketing decision support from a systems design and administration perspective. It will include a brief discussion of software selection and a detailed look at the sizing, testing, tuning, and implementation of the data warehouse. The case study will address such issues as sizing, I/O subsystems, I/O bandwidth, backup issues, performance trade-offs, and day to day operations.

Introduction

When the business systems department said they wanted to look into building a data warehouse they came to the database operations department at AOL and asked us to evaluate the options and choose something that would scale well and provide them with the tool they needed to provide marketing decision support. They only knew how much data they already had and how much it might grow based on projections. We set ourselves to the task of building something that might accommodate what we thought they asked for.

Data Warehouse Software

The first thing that needed to be done was to pick the software to run the data warehouse. After a short search two candidates emerged. Sybase IQ and Redbrick. Sybase IQ was recommended by our local Sybase Inc. representatives and Redbrick was recommended by our local Silicon Graphics Inc. representatives. Oracle was contemplated but was dismissed because it would have been more difficult to test given the lack of a local support system since our relationship with Oracle Inc. wasn't as extensive.

Sybase and Silicon Graphics both had representatives on-site to provide information and support. After testing on demonstration systems, Redbrick was chosen for ease of use and implementation reasons as well as generally good performance. Sybase IQ performed a bit better than Redbrick, in general, during data loads. Redbrick wasn't that far behind and it was ultimately chosen for administrative rather performance reasons. Sybase IQ did not have the ability to add user access to the application without stopping and restarting which would be unacceptable in an environment with constantly changing user requirements and maximum availability requirements.

Disks

Since Redbrick was chosen as the data warehousing software, the demonstration platform was also chosen for testing and implementation. The testing platform was a Silicon Graphics Power Challenge XL with 550 Gigabytes of disk space in a mirrored JBOD configuration for a usable storage area of 225 GB. The machine also had 12 MIPS R10000 CPUs and 2 GB of RAM. This size was fine for testing but not large enough to do any real work on.

The disk space was doubled by swapping the 4.3 GB disk drives for 8.5 GB disk drives. The data was preserved by splitting the mirrored data volumes and replacing half the drives with the larger 8.5 GB drives while keeping the data on the small drives. The data was dumped from the preserved mirror sides on the small drives to filesystems on the new disks and then the new larger filesystems were mounted in the place of the old smaller ones. The remaining old smaller drives drives were replaced with new larger ones and those were mirrored together with the previously replaced drives. The whole process required very little downtime and was made possible by using the XLV volume manager [1] from SGI. This is the machine that was initially put into production.

The total disk space was then doubled again by doubling the number of disks about two months after the start of production. This addition of disks caused performance problems. The machine became very slow and we went back to do further testing on a second machine being built as a companion since the data had once again grown and was now too large to be contained on a single machine.

Detail data was separated from aggregate data in order to accommodate the amount of data required to be online. The testing on the second machine showed

that there were certain I/O limitations inherent in the Challenge XL and also that the RAID configuration for the recently added disks was not optimal.

The major limitations of the Challenge XL were physical space limitations. It can only accommodate 40 SCSI controller connections on its bulkhead which limits any application to 40 SCSI controllers – which would seem like enough until you need very large data sizes for your application along with high performance. More disks could be connected but we found that performance suffered too greatly.

The new machine was set up in a completely different RAID 0+1 configuration that was chosen for performance as well as data protection. Since the amount of data was so large, backups could not be performed right away. We decided that we would just keep two copies of the data at all times with disk mirroring.

The mirroring and striping of the regular SCSI disks was accomplished with SGI's XLV product. The second time the disk capacity was doubled, hardware RAID units from SGI were added and they were configured as a 0+1 as well. The striping of the disks was done to increase I/O performance. The total size of the mixed regular and RAID disk machine is 2.448 TB with 1.224 TB usable due to mirroring. The database size on this machine is approximately 840 GB with the rest of the 1.224 TB or 384 GB used for staging of the raw data. The total size of the all RAID disk machine is 2.72 TB with 1.36 TB usable due to mirroring. The database size on this machine is approximately 870 GB with the rest of the 1.36 TB or 490 GB used for staging of the raw data from the detail machine.

Performance Modeling

Multiple RAID configurations were evaluated in the event that backups would become available. In order to test performance, the applications characteristic read and write behavior needed to be modeled. Since Redbrick does large long running I/Os in 8 KB chunks, it was not that difficult to create a general model of the I/O behavior of the application. Using a simple dd command, the read and write I/O behavior could be adequately modeled. The command used to model write behavior was:

```
% dd if=/dev/zero of=/test/file01 \
     bs=8192 count=100000
```

and the command used to model read behavior was:

```
% dd if=/test/file01 of=/dev/null \
     bs=8192 count=100000
```

These were used to simulate single read and write processes. Redbrick also has the ability to perform parallel reads and writes so multiples of these two commands were linked to specific processors using the built-in processor affinity commands in IRIX 6.2 and rerun to approximate the way Redbrick would run them. See Appendix A for the actual scripts.

The RAID [2] performance data was collected for RAID 0+1, RAID 3, and RAID 5. RAID 0+1 is what is known as a stripe of mirrors in which pairs of mirrored drives are striped for performance reasons. RAID 3 is a group of disks with a drive designated as the parity drive. In a RAID 3 configuration, parity is always written to the same drive while data is striped across the rest in the group. RAID 5 is a group of disks with data and parity striped across all of the drives in the groups in a round robin fashion.

I found RAID 0+1 to perform best for both reads and writes as well as providing the best data protection. It is, however, the most costly configuration. The RAID 0+1 filesystem configuration I used was four disk RAID 0+1 LUNS in groups of eight using four controllers with 128 block RAID stripes and 512 block filesystem stripes. A RAID stripe indicates the size of a data stripe placed on each drive in a RAID group or LUN. The filesystem stripe indicated the size of a data stripe placed on each disk if the filesystem is made up of individual disks or in this case the size of a data stripe placed on an individual LUN since our filesystems are made up of RAID LUNS.

Filesystem throughput was was approximately 80 MB/sec or 10 MB/sec per LUN for a single write and 53.33 MB/sec or 6.67 MB/sec per LUN for four parallel writes. Filesystem throughput was approximately 24.33 MB/sec or 3 MB/sec per LUN for a single read and 44.53 MB/sec or 5.56 MB/sec per LUN for four parallel reads.

The RAID 3 filesystem configuration I used was five disk RAID 3 LUNS in groups of eight using four controllers with a RAID 3 stripe of one and a 512 block filesystem stripe. Filesystem throughput was approximately 48.84 MB/sec or 6.11 MB/sec per LUN for a single write and 52.72 MB/sec or 6.59 MB/sec per LUN for four parallel writes. Filesystem throughput was approximately 23.49 MB/sec second or 2.94 MB/sec per LUN for a single read and 35.68 MB/sec or 4.46 MB/sec per LUN for four parallel reads.

RAID LEVEL	1W	4W	1R	4R
RAID 0+1	80 MB/sec	53.33 MB/sec	24.33 MB/sec	44.53 MB/sec
RAID 3	48.84 MB/sec	52.72 MB/sec	23.49 MB/sec	35.68 MB/sec
RAID 5	30.32 MB/sec	32.91 MB/sec	14.39 MB/sec	42.93 MB/sec

Table 1: Performance summary.

The RAID 5 configuration I used was five disk RAID 5 LUNS in groups of eight using four controllers with a RAID 5 stripe of 128 blocks and a filesystem stripe of 512 blocks. Filesystem throughput was approximately 30.32 MB/sec or 3.79 MB/sec per LUN for a single write and 32.91 MB/sec or 4.11 MB/sec per LUN for four parallel writes. Filesystem throughput was approximately 14.39 MB/sec or 1.80 MB/sec per LUN for a single read or 42.93 MB/sec or 5.37 MB/sec per LUN for four parallel writes. The results are summarized in Table 1.

From the tests I found that RAID 3 had comparable write performance to RAID 0+1 but it was approximately 20% slower on reads than RAID 0+1. RAID 5 had comparable read performance to RAID 0+1 but it was approximately 38% slower on writes. I decided RAID 0+1 was the only viable option since the slower writes of RAID 5 would extend load times of the database unacceptably and the slower read times would cause user queries of the data to run longer which could potentially cause problems with service requirements. RAID tests with other configurations were done with these three RAID levels. They are not reported here since they were used to optimize the stripe sizes for each RAID level and cache read/write size distributions since the hardware RAIDs have built in caches for further speed up of reads and writes for general performance improvements.

Backups

Backups were initially planned using existing DLT4000 drives in robot tape changers on the network. The size of the data quickly outgrew the capacity of the DLT robots and the network. Alternate solutions were sought while the RAID 0+1 was the only method of data protection. Adding to the problem was the limited number of controllers left for attaching disk devices after disk expansion and the relatively short backup window.

Of the limited total number of SCSI controllers available on the Challenge XL, 36 had already been used for the disks that were attached. Since only four SCSI controllers were available for backup devices and approximately 1 TB needed to be backed up in a six hour window, we needed to find high bandwidth backup devices.

The devices we tested were multiple DLT 7000 robot tape changers with multiple drives on each SCSI controller and multiple DLT 7000 robot tape changers connected to a remote machine with a HIPPI network for connectivity. An AMPEX 812 DST was considered but physical size and lack of software support caused us to rule it out.

The directly attached DLT 7000 robots worked quite well and were able to sustain 7 MB per second throughput per drive when they were directly attached to a machine via fast-wide differential SCSI connections. Three drives per SCSI channel were used to get maximum throughput per SCSI channel. The backups of the smaller machine in this configuration took 3.58 hours. This backup included database and raw data staging areas.

Dedicating three large, expensive tape robots to a single machine where they would be used at most only one third of the time seemed like a bad idea, so we looked for other options to move data off the machines at high speed. We spoke to our local SGI reps and they suggested we investigate HIPPI [3] with an interesting twist called Bulk Data Services [4], which is a high performance NFS add-on used to take advantage of the high bandwidth of HIPPI by bypassing the buffer caches on the local and remote machines.

With BDS, instead of doing a normal NFS transaction, the buffer caches are ignored on both sides and the data is retrieved via direct I/O from the machine where it resides and it is passed via an 'aligned network send' straight into memory on the remote machine where the pages are then flipped to the application. BDS enables long running streaming IO's like the kind we were doing during backups to bypass much of the overhead of NFS, enabling much greater performance.

The filesystems that were to be backed up were exported via NFS on a warehouse machine with BDS installed to a machine that also had BDS installed and several DLT 7000 tape robots as well. A slight software modification was required in the backup software to take advantage of BDS as well. After all of those issues were addressed, backups were run and throughput was measured to be approximately 5 MB/sec per tape drive with 10 drives running simultaneously. The aggregate throughput of the HIPPI link between machines was approximately 50 MB/sec. This allowed the large warehouse machine to be backed up fully in five hours.

The HIPPI/BDS configuration fit in with our backup time requirements and allows us to use the devices for backups of more than one machine which provides a savings in cost and amount of hardware needed. The HIPPI/BDS configuration is the one we will finally be using for backups.

Nearline Storage

In addition to the backups to tape, we decided that some other more near line area was needed to store old but not necessarily unneeded data. Redbrick has the ability to unload tables of data in a format that can be easily reloaded and that also takes up less space than the original tables. An NFS server seemed ideal for seldom accessed data that should be held onto but didn't need to be online. A Sun Ultra 2 was set up with 200 GB of disk space contained in a Sun Storage RSM219. Two 100 GB filesystems were mounted on each of the data warehouse machines for the purpose of holding this seldom used data. The backups and the NFS server allowed us to to have protection from

catastrophe as well as an easy to use near line data space for old warehouse data.

Administration

Certain compromises were made to give adequate performance and still keep administrative headaches to a minimum. Redbrick's consultants initially recommended filesystems no larger than 2 GB, since no single Redbrick file can be over 2 GB in size. These 2 GB file systems can be arranged to give increased performance through striping between them but there would have been several hundred filesystems in a 1 TB data warehouse. We decided to compromise by using a few large filesystems but still striping the files themselves across multiple filesystems and carefully avoiding using the same filesystem for two things at the same time during a single operation.

When loads are done, for instance, the loaded data is read out of single filesystem and written out to multiple filesystems consecutively. The consecutive nature of the data writes is a Redbrick limitation but it simplifies the process of ensuring that no SCSI controller bandwidth contention occurs. We can take care to use a filesystem that uses one set of SCSI controllers for reading staged data and a completely different set of SCSI controllers for table writes.

Redbrick is generally simple to get set up and it requires little adjustment to automate its startup and shutdown. The server needs to be started as root which is inconvenient but can be accomplished on start up and shutdown with an init script in the appropriate place. sudo access can be given to the Redbrick install uid for starting and stopping the server at times other than startup and shutdown of the machine. This reduces the total administrative load from Redbrick software. Checkpoint and restart software is also being tested for use with long running jobs to avoid redoing work in the event of an unforeseen machine stoppage such as reboot or filesystem overflow. Initial testing is promising but it is still too soon to tell and our testing continues.

Performance Tuning

Certain kernel parameters require tuning to get better performance from Redbrick. The filesystem cache should be as large as possible and as much RAM as possible is indicated. The fraction of RAM allowed to be allocated to a single process should also be maximized since Redbrick keeps its working data set in memory until it is written out. This can be several hundred megabytes depending on the size of the data set.

The fraction of RAM available to user processes is 75% by default under IRIX 6.2 and we found that making this fraction 90% helped performance without hurting the normal operation of the machine due to the rather large installed memory sizes. In the 4 GB RAM machine, 90% usage still leaves 400 MB of RAM for system usage, which is plenty. The number of open file handles allowed by default was also adjusted upward since Redbrick does many file I/O's and tends to keep file handles open for a long time while doing long queries or loads.

Conclusion

During the implementation of this data warehouse we learned a tremendous amount about I/O performance tuning for large scale data volumes. Additionally, we found many ways to work around and live with the limits of current machines as well as making suggestions for improvements in certain subsystems. This work has also helped us plan for the growth of the application and its supporting hardware.

The RAID benchmarking and I/O tuning has also helped to create a general set of expected behaviors for the disk subsystems of all our machines. This will help us to plan more efficiently in the future for all projects. Growth of the data warehouse continues and that has caused us to look for newer larger more powerful machines for processing the warehouse data. We are hoping to implement a new solution based on fibre channel-arbitrated loop disks attached to an SGI Origin 2000 which will allow the win-win of a larger data space while increasing performance as well mostly due to the advantages of the FC-AL disks and the improved bandwidth of Origin.

Author Information

Dan Pollack was introduced to UNIX in 1988 and has been a System Administrator of one sort or another since 1990. He has worked in the financial, government and online service industries. He currently resides at America Online Incorporated in Reston, Virginia where he is a Senior System Administrator. He can be reached via email at dpollack@aol.net or via US mail at 12100 Sunrise Valley Drive Room 1Q03 Reston VA, 20191

References

[1] "IRIX Admin: Disks and Filesystems," *IRIX 6.2 Reference Manual,* Document Number: 007-2825-001, Silicon Graphics Inc, 1996.

[2] "RAID FAQ," *Frequently Asked Questions about RAID Levels*, http://www.recoverdata.com/raidfaq.htm .

[3] *Gigabit Networking*, Craig Partridge, Addison-Wesley Publishing, 1994, Chapter 7.

[4] "Getting Started with BDSpro," *IRIX 6.2 Reference Manual*, Document Number: 007-3274-001, Silicon Graphics Inc., 1996.

Appendix A

Single Write Script

```
#!/bin/sh
timex dd if=/dev/zero of=/test/fs1/file01 bs=8192 count=100000
```

Single Read Script

```
#!/bin/sh
timex dd if=/test/fs1/file01 of=/dev/null bs=8192 count=100000
```

Four Writer Script

```
#!/bin/sh
runon 1 timex dd if=/dev/zero of=/test/fs1/file01 bs=8192 count=100000 &
runon 2 timex dd if=/dev/zero of=/test/fs2/file01 bs=8192 count=100000 &
runon 3 timex dd if=/dev/zero of=/test/fs3/file01 bs=8192 count=100000 &
runon 4 timex dd if=/dev/zero of=/test/fs4/file01 bs=8192 count=100000 &
```

Four Reader Script

```
#!/bin/sh
runon 1 timex dd if=/test/fs1/file01 of=/dev/null bs=8192 count=100000 &
runon 2 timex dd if=/test/fs2/file01 of=/dev/null bs=8192 count=100000 &
runon 3 timex dd if=/test/fs3/file01 of=/dev/null bs=8192 count=100000 &
runon 4 timex dd if=/test/fs4/file01 of=/dev/null bs=8192 count=100000 &
```

Shuse At Two: Multi-Host Account Administration

Henry Spencer – SP Systems

ABSTRACT

The Shuse multi-host account administration system [1] is now two years old, and clearly a success. It is managing a user population of 20,000+ at Sheridan College, and a smaller but more demanding population at a local "wholesaler" ISP, Cancom. This paper reviews some of the experiences along the way, and attempts to draw some lessons from them.

Shuse: Outline and Status

Shuse [1] is a multi-host account administration system designed for large user communities (tens of thousands) on possibly-heterogeneous networks. It adds, deletes, moves, renames, and otherwise administers user accounts on multiple servers, with functionality generally similar to that of Project Athena's Service Management System [2].

Shuse uses a fairly centralized architecture. All user/sysadmin requests go to a central daemon (*shused*) on a central server host, via *gatekeeper* processes on that host, which handle network interaction and authentication, and insulate the daemon from user misbehavior. *Shused* itself handles all database updates, regenerates derived files like the *passwd* file as necessary, and makes calls to various helper processes. *Shused* is single-threaded using an event-loop organization; long-running operations (e.g., updates to "slave" servers) are done by an auxiliary daemon (*shuselace*, which talks to *shused* using roughly the same command interface employed by users), so that *shused* itself is not tied up and unresponsive during such operations. *Shuselace* performs operations on a slave server by invoking (via *telnet* and *inetd*) a *shusetie* process there.

Shused keeps the entire user database in memory for fast response (RAM is cheaper than database packages). Crashproofing, initial startup, and emergency manual changes require a copy of the database on disk, but the presence of the memory copy allows optimizing the disk copy for quick updates rather than rapid bulk access. The disk copy is stored as one file per user, using a simple text format with each line containing a field name and a field value.

Shuse was written essentially entirely in Expect [3, 4] (an extended version of Tcl [5, 6]), a decision we have not regretted. A few tiny auxiliary programs written in C do jobs that are not feasible in Expect, and there are some shell files around the periphery as well. Performance has been quite satisfactory after a few early problems were resolved. Maintenance and enhancement have been greatly eased, and portability has been trivial.

After some trying moments early on, Shuse is very clearly a success. In mid-autumn 1996, Sheridan College's queue of outstanding help-desk requests was two orders of magnitude shorter than it had been in previous years, despite reductions in support manpower. Favorable comments were heard from faculty who had never previously had anything good to say about computing support. However, there was naturally still a wishlist of desirable improvements.

At around the same time, ex-Sheridan people were involved in getting Canadian Satellite Communications Inc. ("Cancom") into the ISP business as a "wholesaler" ISP, supplying connectivity and resources to a network of retail dealers in mostly-remote areas. They decided they wanted to use Shuse.

Evolution

Cancom's use of Shuse required tracking slightly different information and generating slightly different outputs. This exposed a fair number of Sheridan-specific assumptions, some of which were easier to cure than others.

Merely adding new fields to the database was fairly trivial, thanks to the early decision to adopt a highly extensible format.

Changes to the Shuse code were also required, however, and since the Sheridan version was inevitably evolving in parallel, periodic code merges have been required ever since. This has been a bit tedious but not fundamentally difficult; we're still discovering which things need to be parameterized so that configuration files can customize them to customer needs. We don't think we could reasonably have anticipated most of them; indeed, some of the early attempts at such anticipation have gone unused.

Assorted additional facilities also had to be added, such as a Shuse implementation of the *poppassd* interface that lets non-shell users change their passwords. (This was originally done for Cancom, but turned out to be of considerable interest to Sheridan as well – a pattern that has held for a number of the changes.) We initially tried to make the usual freeware *poppassd* implementation talk to Shuse, but after a

number of problems (including one security breach via a core dump!), we gave up in disgust and wrote our own in Expect. It turned out to be shorter and much more robust and versatile than the original C code (which is basically trying to do an Expect-like job without the proper tools).

A more substantial user interface that also had to be added was a menu-oriented interface for Cancom's dealers, so that routine account administration could be delegated (subject to appropriate restrictions) to them. Original ideas of GUIs based on Tk [6] had to be shelved in favor of a very simple text-based menu system because of a tight schedule, unpredictable variations in user equipment, and the limitations of "long thin" communications links.

The dealer interface is not glossy and elegant, but it has worked out well. We were worried about response time, especially given those communications links, but the dealers have been so happy at being able to do their administration themselves – getting results in seconds instead of hours or days – that a bit of slowness hasn't yet elicited any critical comment. The one real blemish of the current design is that the dealer program knows too many things that *shused* also knows, so any change to such things has to be made in several places. Fixing that will require more complex provisions for user interfaces to obtain such information from *shused*, perhaps as downloaded Tcl code – a promising approach, but also a complicated one.

The text-based dealer interface naturally speaks English. This being Canada, soon after the dealer interface went into operation we got asked "what about a French version?" Intense distaste for the thought of maintaining two separate versions of the same code, plus a slight possibility of needing more languages in future (Cree has been mentioned...) led us to invent a more general solution: a message-catalog system for Tcl [7]. This hasn't seen enough use yet for a good evaluation, but it seems adequate to do the job.

We note that although it was originally envisioned that there would be multiple user-interface programs talking directly to *shused*, in practice all interfaces to date have been built on top of the *shusedo* program, which simply sends a single command to the daemon and outputs the response. A simple command-line interface like this lends itself well to the construction of more complex and more interactive interfaces; the reverse is *not* true.

Decentralization

The original Sheridan version of Shuse relied very heavily on NFS – in particular, it shared the */usr/local/shuse* tree that way – and distributed the *passwd* file (etc.) using NIS/YP. Cancom did not particularly want to do either, mostly for reasons of security. Sheridan wanted to continue using NIS but was having second thoughts about having an important administrative area widely NFS-mounted. Considerable effort was needed to adapt Shuse to a more loosely-coupled environment.

We had to make a concerted search for places where Shuse components explicitly or implicitly relied on shared filesystems or NIS, and fix them all. Inevitably, one or two were discovered only after the code was put into service. (One particular complication was that Sheridan was still running with a shared */usr/local/shuse* temporarily. It was not enough that the new code work correctly with a non-shared */usr/local/shuse*: it had to work whether */usr/local/shuse* was shared or not, and NFS's odd treatment of *root* caused minor difficulties here.)

The *telnet* connection which *shuselace* uses to give orders to *shusetie* is unsuited to bulk data transmission, which originally was done via NFS file sharing. Bulk data transfers are now done by FTP, using Expect to drive the Unix *ftp* utility.[1] We picked FTP because it was already installed and the networks involved are reasonably well controlled; a future shift to cryptographic authentication, e.g., via *ssh/scp*, is being considered for Cancom in particular.

Finding and fixing the more subtle dependencies on shared files and NIS was tedious, but it has had useful side effects. For example, fixing Shuse's password changers to consult *shused* for the old encrypted password, rather than doing their own accesses to the *passwd* file or invoking *ypmatch*, has also made them compatible with shadow password files.

The original Shuse updated the *passwd* file by simply generating a copy of the new file in a known location, whence a *cron* job regularly picked it up and shoved it into NIS. Without NIS, Shuse had to do its own updates of the *passwd* file on slave servers, which is harder than it sounds because of the shortage of decent programming interfaces for this.[2] We ended up telling *vipw* to use *ed* as its text editor, and driving *ed* via Expect. This had a few problems of its own (such as discovering that we could easily overdrive a BSD/OS pseudo-tty to the point where it would lose characters), but with careful checking and some judicious use of some of Expect's more obscure features, in the end it worked. (The pseudo-tty overdriving problem, in particular, was solved by insertion of a 1 ms delay every 40 characters, which is easy with Expect.) The resulting code is so paranoid that it has

[1] The *ftp* program is another one of those wonderful utilities which *knows*, by God, that it is running interactively, and completely neglects to provide any sort of reasonable *programming* interface. Fortunately, Expect deals with this reasonably well, at the cost of some tedious experimenting to uncover all the likely interactive messages.

[2] As with FTP and quotas, the *passwd*-file user interfaces are a disgrace to Unix, since they typically insist on running interactively and cannot be programmed without resorting to Expect.

caught various unanticipated problems (e.g., *vipw* running out of disk space).

The lack of shared filesystems necessarily means replicating the Shuse software on the various hosts. This has been somewhat error-prone, and we've been working on reducing its problems. We've been progressively eliminating shared control files that have to be kept consistent: about the only ones remaining are authentication keys and Shuse's configuration files. Some of the eliminated files never really had to be shared; others are necessary but are small enough to be transmitted each time. We've also been making an effort to reduce the number of code files that have to be present on a slave server, although there are limits to this.

More fundamentally, Shuse is now taking on the responsibility of updating the remote copies of itself! This is arguably a re-invention of the wheel, since existing software packages like *rdist* [8] can handle this sort of thing. However, one of the customers wanted it and was willing to pay for it, so we did it. It was quite straightforward, using FTP for data transfers, except for the need to make careful provision for backing out of an update that happens to break the slave-server side of the software.

Since the remote-update facility now exists, it's also being used as a substitute for other inter-machine propagation mechanisms. For example, Cancom is using it to propagate updates to */etc/group*.

Could all this effort have been avoided with greater forethought about loosely-coupled systems? Probably most of it, yes, but there were good reasons why it wasn't done that way the first time. For one thing, priorities have changed with experience: some of the original approaches had apparent advantages that have not proved significant in practice.[3] For another, the new mechanisms have added substantial complexity in places, and time constraints weighed heavily in the early development of Shuse [1].

Internal Cleanup

Every time we've put effort into cleaning up and generalizing Shuse's innards, we've regretted not doing it sooner. Many things have become easier this way; many of the remaining internal nuisances are concentrated in areas which haven't had such an overhaul lately.

The most fundamental area of internal change has been the protocols used to request updates on the slave servers. The original Shuse design envisioned a very simple and appealing approach: *shused* would distribute a description of the way things were supposed to be, and the slave servers would compare this to the actual situation and do any changes needed to make reality match the description. This has the advantage of being extremely robust in general, and completely crash-proof in particular: no matter who crashes and when, if everybody is eventually up for long enough, the slave servers will synchronize with *shused*'s current opinion on how things should look.

Unfortunately, too many details of real account maintenance did not fit this description-based model very well. There were a few early signs of difficulty, but the real killer was the need for coordinated activity by two slave servers when moving a user from one server to another.

A number of difficulties were cleared up by breaking down and conceding that some operations would simply have to be done in tight lockstep, with *shused* retaining a to-be-done list and checking items off as they were completed. Making such operations crash-proof is not fundamentally difficult, given the facility for planting "at commands" (to be executed at specific times) in a user's database entry, although the details unfortunately ended up being rather complex.[4] This was first done for user moves, and has since been extended to renames and quota settings. User deletion, implemented originally using the description-based model, will probably move to a lockstep implementation.

On the other hand, serious consideration is being given to moving back towards the description-based model for some things. In particular, disk-quota updates are currently done as lockstep operations, but this falls down badly in situations like restoring a user filesystem from a backup. Quotas probably *should* be handled with the description-based model, and we're going to look harder at this.

The moral we draw from this is that one should not overlook the need for feedback: the key oversight in the original description-based model was that even though *shused* is always leading and the slave servers are always following, some operations do require *shused* to know exactly when a change takes place on a slave server.

Much of the other internal-cleanup work has been focused on what might be called improving the software engineering of Shuse's innards. Useful facilities have been generalized and encapsulated to make them easier to use for multiple purposes. Shared knowledge has been eliminated, e.g., by making data-transmission formats self-describing. Bright ideas that turned out to be mistakes have been ripped out and replaced by simpler approaches. None of this should

[3] For example, we'd originally hoped to avoid having to use shared secrets for authentication, by relying on permissions of shared directories instead. Shared secrets proved necessary for other reasons.

[4] Getting a single lockstep operation done in a fully crash-proof way involves four or more at-commands interwoven in a pattern resembling a database two-phase-commit protocol. The internals documentation calls this process "The Dance of the At-Commands."

really require comment, except that it's so seldom actually *done* on real software. We've found that effort spent on this typically pays off handsomely, by making later changes easier.

Administration

Although it would be nice if automated-sysadmin software didn't itself require administration, it does. The interfaces used for this are sometimes skimped on. Shuse has needed improvements in several areas of its administrative interfaces.

One area that fortunately *hasn't* needed much improvement is operational robustness. There is an obvious vulnerability in a single central server process running on a single central server host: what happens if the process or the host crashes? We originally decided that it was better to spend effort on eliminating process and host unreliability than to devise elaborate distributed protocols to cope with it. This decision has been amply vindicated.

Making the host reliable hasn't required much effort. Making the *shused* process reliable did take some. In particular, early development versions of *shused* died whenever their innards signalled an error, on the theory that it might have caused corruption of the database. This got fixed before entry into production: while there is *potential* for database corruption, most real error signals denote nothing more fundamental than a minor bug in the particular request being executed, and logging the problem and carrying on is vastly preferable to falling over dead.

Given this, *shused* has proved reliable enough that we've done nothing at all about automated recovery from catastrophic failures: if it happens, the staff notice and restart the daemon. While new versions of *shused* have occasionally fallen over suddenly, once such last-minute problems are resolved, reliability has been high. The *shused* process running at the time this is being written, in early September, has been running since a development-induced restart at the end of July.

A related but more subtle problem is that early versions of *shuselace* occasionally died under mysterious circumstances. This was much more subtle than having *shused* die, because *shused* was still handling interactive commands and database updates properly, only the updates didn't propagate out to the slave servers. Although the problem hasn't happened recently, *shused* now guards against this and other *shuselace* failures by "pinging" *shuselace* regularly and complaining if there is no response. Again, the problem hasn't been frequent enough to justify automated handling (which would be slightly awkward due to implementation details); the fix is to shut down and restart *shused*.

An area that was, for a long time, rather less satisfactory was trouble reporting. The original *shused* design lacked any orderly way of reporting difficulties to the sysadmins. Theoretically one could regularly inspect the log file, but in practice this didn't get done very much.

The problem was greatly aggravated with the arrival of lockstep operations like renaming a user, where the need for slave-server cooperation means that actual execution of the operation may be arbitrarily delayed pending availability of the necessary slave servers. The command executed by the user merely queues up the operation, and originally there was no systematic way of reporting success or failure of the ultimate execution. This was particularly troublesome because such operations *usually* succeed quickly, and so one gets out of the habit of checking on them.

After a period of just muddling along, this got fixed in the obvious way: the results of delayed operations, and independently-discovered indications of trouble, are reported by mail messages. This wasn't quite as simple as it looked, because an administrator who moves 500 users (not at all unusual in a user population of 20,000+) does not want to get 500 separate mail messages reporting success. An ambitious design for merging similar messages was thought out but shelved (although there are implementation hooks for it) in favor of a simple timeout mechanism, which just holds onto non-urgent reports briefly in hopes of being able to send more than one report in a single message. The details are still being tuned, but this seems generally adequate. We should really have thought about this, and done it, at the time the first delayed operations appeared (if not earlier).

While the *shused* log file (which incorporates log entries sent in by other components of the software, like *shusetie*) is useful for debugging, it's large, and less than ideal when it comes to answering questions like "who last changed user *xyz*'s disk quota?" A little bit of historical information is kept in user database entries as it stands – for example, there is a timestamp field identifying the time and instigator of the last password change – but this isn't always adequate. Work is now underway on a general facility for holding a configuration-specified amount of command history in each user database entry, so that information on at least the recent changes will be *conveniently* available when needed.

Conclusion

Shuse continues to evolve, partly because we're still learning what it needs to do. Expanding operations to a second user community has made a lot of learning happen fairly suddenly. We're also belatedly recognizing things we should have noticed a while back, such as the need for better administrative interfaces.

Acknowledgements

Sheridan College in general, and Cheri Weaver in particular, got Shuse started. John Barber of Sheridan and Doug Berry of Cancom have supported its

continuing evolution. A number of people, most notably Trevor Stott and Doug Berry, have used Shuse at length and have been patient as problems were found and fixed.

Availability

Bad news: it's still not freely available, alas. Sheridan continues to be interested in the possibility of commercial marketing, although nothing organized has happened yet. If you're interested in getting Shuse, contact the author, and he'll pass your inquiry on to the right people.

Author Information

Henry Spencer is a freelance software engineer and author. His degrees are from University of Saskatchewan and University of Toronto. He is the author of several freely-redistributable software packages, notably the original public-domain *getopt*, the redistributable regular-expression library, and the *awf* text formatter, and is co-author of C News. He is currently immersed in the complexities of implementing POSIX regular expressions. He can be reached as henry@zoo.toronto.edu .

References

[1] Henry Spencer, "Shuse: Multi-Host Account Administration," *Proceedings of the Tenth Systems Administration Conference (LISA X)*, September 1996 (Chicago), Usenix Association 1996.

[2] Mark A. Rosenstein, Daniel E. Geer, & Peter J. Levine, "The Athena Service Management System," *Proceedings of the Usenix Technical Conference,* Winter 1988 (Dallas), Usenix Association 1988.

[3] Don Libes, "Expect: Curing Those Uncontrollable Fits of Interaction," *Proceedings of the Usenix Technical Conference,* Summer 1990 (Anaheim), Usenix Association 1990.

[4] Don Libes, *Exploring Expect*, O'Reilly & Associates 1995.

[5] John K. Ousterhout, "Tcl: An Embeddable Command Language," *Proceedings of the Usenix Technical Conference,* Winter 1990 (Washington), Usenix Association 1990.

[6] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley 1994.

[7] Henry Spencer, "Simple Multilingual Support for Tcl," *Proceedings of the Fifth Annual Tcl/Tk Workshop,* July 1997 (Boston), Usenix Association 1997.

[8] Michael Cooper, "Overhauling Rdist for the '90s," *Proceedings of the Usenix Technical Conference,* Winter 1987 (Washington), Usenix Association, 1987.

A Web-Based Backup/Restore Method for Intel-based PC's

Tyler Barnett – Lexmark International
Kyle McPeek – Aerotek Inc., on behalf of Lexmark International
Larry S. Lile – Aerotek Inc.
Ray Hyatt, Jr. – Aerotek Inc.

ABSTRACT

This paper describes two similar user-initiated methods to backup and restore PC workstations over an ethernet network. The first method concerns "PC Hardware All Alike," or the backup and restoring of sets of identical disk images across many identical systems. The second method is the backup and restoring of "PC Hardware All Different," which manages unique disk images on individual systems.

This paper describes the installation and customization of a "controlling" unix partition on each client system PC platform. The purpose of the unix partition is to quickly backup and restore Windows and OS/2 disk images on a separate partition on the client system hard drive.

The web page interface is used to schedule the image backups and restores. The unix server that supports the backup and restore process is described, along with the scripts that control the backup and restore process. Usage statistics are generated that can quantify the cost benefits of the backup and restore method.

Background

Lexmark's core business is printing solutions, which are primarily inkjet and laser printers, and the product line changes rapidly. Testing is required of a large amount of printer drivers and networking utilities, all of which are translated to numerous foreign languages. This software testing requires the testers to have at their disposal a client PC platform capable of Win31, Win95, WinNT, and OS/2 to support all the versions of drivers and utilities.[1] This testing mission is loosely referred to as NLS, or National Language Support. From now on, "NLS" is equivalent to the backup and restore mission of "PC Hardware All Alike."

The Problem

Once a printer driver or network utility is installed and tested on a PC, further installation of yet another revision of drivers or utilities would not reflect the customer situation. The "now corrupt" system could possibly mask installation problems, or cause other problems due to the old drivers or utilities still on the hard drive. Based on experience, it is virtually impossible to reliably and economically "uncorrupt" a system by removing all possible installed DLL's, EXE's, DRV's, and help files. The real solution is providing a quick method of installing and configuring a fresh OS on demand.

Two years ago, our organization was directed to support over 20 foreign languages on all products. Installation of a foreign language operating system contains its own challenges, and usually the tester is not conversant in that language. We were asked to setup the infrastructure for an NLS Lab that would support these languages during on-site visits by teams of reviewers from various foreign countries. These "native" reviewers would use these systems during their on-site visit.

A simple system was needed that foreign reviewers could use, without getting the sysadmin involved during an on-site workshop. This web backup and restore method allows the reviewer to quickly restore any previously installed language/OS combination, and be assured it is correct. This also gives our Test team the ability to quickly return the client system to a known state at any time.

Supported Hardware/Software

A decision was made to only support PC's with SCSI and ethernet. Token-ring network cards and Microchannel systems are not supported, because neither is supported by FreeBSD.[2] Since FreeBSD can be installed everywhere without additional licensing fees, and had proven reliable in several other large projects, it was our choice for this project.

This backup and restore method could be easily adapted to other versions of unix by making changes in the scripts for different device names, and other

[1] Printer drivers and utilities are installed on the client PC during normal printer installation.

[2] http://www.freebsd.org/handbook/handbook11.html.

minor changes. Users familiar with unix and perl should be able to use our scripts with minimal effort.

History

The first NLS requirement was for UK English, French, German, Spanish, and Italian to be each loaded onto five separate P-75 systems. Only Windows 3.1 and OS/2 2.1 were supported. OS/2 Boot Manager was used to switch between bootable partitions. Rebuilding the systems was done manually. This was deemed too labor intensive after one NLS workshop.

The first try at duplicating SCSI drives used a small C program with DOS BIOS calls to read a 512 byte sector, and write it onto an identical drive on the SCSI chain. This proved unusable because a 2 GB drive took over 24 hours to duplicate. This was a dead end, but directly copying the drive was still in our minds.

A bootable DOS diskette was built that logged onto an OS/2 server as an IBM LAN Manager client.[3] It allowed an entire 200 MB C: partition to be backed up onto a server drive using the DOS XCOPY program, and restored as well. Several language workshops using Win31 and OS/2 were conducted using this method.

The number of client systems was increased to 10, two for each language. The ethernet hub was discarded in favor of a 10 MBit ethernet switch, because packet collisions during multiple restores stretched a restore to several hours.

The matrix grew to six languages with the addition of Brazilian Portuguese, and three operating systems with the addition of Windows 95. The introduction of Windows 95 broke our XCOPY method with its long filenames and registry.

We purchased a commercial solution called CSCDUPE[4] that would read/write sectors from a SCSI drive to/from a DOS filehandle on a NETBIOS server. This software was marketed primarily for burning CD-ROM's, and requires the loading of an ASPI SCSI DOS driver in CONFIG.SYS, and running CSCDUPE on top of it.

Entire 200 MB disk images were saved without compression on several 2 GB server drives. A restore of a 200 MB C: partition took 45 minutes across a 10 MBit ethernet segment on a P-75 using an Adaptec 2940 and a SCSI-2 drive.

Several more language workshops (including Win95 testing) were performed with this CSCDUPE method. However, CSCDUPE is entirely menu-driven and prone to input errors. A better method was needed that freed the sysadmin from this task.

As an experiment, a FreeBSD[5] boot diskette was constructed that NFS mounted a FreeBSD file server. Then the dd command was successfully used to backup and restore a DOS partition from one client system to another, with the image stored on the FreeBSD server. Several iterations of this method were explored, including attempts at BOOTP using a boot EPROM on the ethernet card, and a DOS boot diskette that invoked a BOOTP of FreeBSD.[6]

The final method implemented was a separate 200 MB FreeBSD partition created on each client system, while Booteasy[7] was used to boot either FreeBSD or the DOS partition with the target OS/language. This method does not require boot diskettes.

An ethernet switch was installed with a 100 MBit uplink to the server, which provided 24 client 10 Mbit downlinks.[8] The Concatenated Disk Driver (ccd)[9] on the FreeBSD server striped data on two wide SCSI drives to increase throughput, and gzip/gunzip was used on the client system to compress/decompress the images stored on the server.

The final benchmark on P-166 client systems using Adaptec 2940UW, a wide SCSI drive containing a 200 MB DOS partition, and 10 Mbit ethernet was 7.5 minutes for a restore.

The biggest improvement was the web page interface that allowed scheduling of image backups and restores by the user, without getting the sysadmin involved. This web page is served by Apache[10] on the FreeBSD server holding the images.

NLS User Perspective of a Backup

The NLS tester schedules a backup on the Web page using a few mouse clicks to select their hardware system name, desired operating system, and foreign language. A password is needed to perform a backup, which keeps an inexperienced user from overwriting existing backup images. The Web page instructs the user to boot their client system into FreeBSD Booteasy, which briefly asks the user for two choices: "`F1 for BSD`," or "`F2 for DOS`." After pressing F1, there is no further user interaction. The user will later receive a message from FreeBSD on the client system that the backup is complete.

NLS User Perspective of a Restore

A restore is the same as the backup process, but without needing a password. The user schedules the desired restore on the same Web page, and boots the client system into FreeBSD using "`F1 for BSD`."

[3] We had to use 2.88 MB diskette drives and diskettes to hold the OS/2 boot diskette and networking.

[4] http://www.corpsys.com.

[5] http://www.freebsd.org.

[6] http://www.freebsd.org/tutorials/disklessx/disklessx.html.

[7] http://www.freebsd.org/tutorials/multios/multios.html.

[8] http://www.3com.com/products/switches.html. The ethernet switch we use is called a Linkswitch 1000.

[9] http://www.freebsd.org/handbook/handbook50.html.

[10] http://www.apache.org.

The user will later receive a message on the client system that the restore is complete, at which time the user reboots the client system, presses "F2 for DOS," and boots into their desired Windows or OS/2 environment. This provides a guaranteed clean install in a fraction of the time normally taken by ordinary media installation. Currently, a 400 MB image backup or restore takes less than 11 minutes.

NLS Client System Description and Configuration

Our current platform is a Dell GxPro Pro-200, with Adaptec 2940UW[11] SCSI and ultra-wide drives because we wanted the fastest drives possible at the time. The FreeBSD kernel supports Adaptec tag-queuing for better performance.[12] The internal 3Com 905X ethernet is used at 10 Mbit, although 100 Mbit is available on the Dell platform.

The heart of the backup/restore method is the installation of FreeBSD in a separate physical partition on each client system hard drive. This partition has been standardized at 200 MB, with root, var, and swap of 32 MB each, and the remainder devoted to usr.

The SCSI drive is first low-level formatted with the Adaptec ROM utility, and then a DOS diskette is booted to fix any network card EEPROM settings. A FreeBSD boot diskette is loaded, and we FTP install a "Kernel Developer" system, which consists of all binaries (except Xfree86) and all kernel sources. The Booteasy boot manager is also installed.

Once the FreeBSD operating system is booted for the first time, a custom kernel can be configured based on the GENERIC configuration file supplied.[13] This custom kernel is needed for activating the Adaptec tag-queuing support, and removing device drivers that don't need to be probed, and waste time during boot. Here are the lines in the kernel configuration file necessary to be added or changed; see Figure 1. The FreeBSD sysconfig file is modified to make the client system an NFS client, and the hostname, and IP addressing is configured; see Figure 2. Above is an example of an Intel Pro/100B ethernet card entry.

An entry is made in fstab to NFS mount the FreeBSD server that supports the client:

```
server:/array /array nfs rw 0 1
```

An entry is made in rc.local that checks for any backup or restore requests at the very end of booting, and tells the user what it's doing:

```
[-f /array/bin/task ] && \
            /array/bin/task
```

If the script task does not find any backup or restore work to be done, `task` does some housekeeping (explained later), exits normally, and the familiar `login:` prompt appears on the client system.

If the client system is booted without a scheduled backup or restore, then it needs to be shutdown cleanly sometime later. To address this problem, a user id called "shutdown" is created. Then the "shutdown" id is setup to run a simple shell script:

```
shutdown:*:0:0:Quick Shutdown:
        /home/shutdown:/sbin/scram
```

To shut the client system down cleanly, the users are told to login with the id "shutdown," and the script `/sbin/scram` is run at login time:

```
#!/bin/sh
/sbin/shutdown -r now
sleep 60
```

Finally, the FreeBSD root password is set to keep users from changing anything, the client system BIOS setup password is enabled, and the "CTRL-A" boot message is disabled on the Adaptec ROM setup menu.

Once the client system is configured with FreeBSD, then a DOS diskette is booted, and DOS FDISK is used to set the desired size of the user partitions (C: and D:), and both are FAT-formatted. At this point, the completed system is ready to be duplicated on the rest of the NLS system inventory.

In summary, the client system hard drive is organized as below:

[11]http://www.adaptec.com/work/AHA2940UWOF.html.

[12]http://www.freebsd.org/releases/2.0.5R/notes.html. Only release 2.0.5 and beyond has tag queuing.

[13]Found in /usr/src/sys/i386/conf. Copy GENERIC to your own file, and edit it to suit.

```
config      kernel  root on sd0     # default was IDE
controller  ahc0                    # Adaptec 2940 driver
options     "AHC_TAGENABLE"         # Adaptec support
options     "AHC_SCBPAGING_ENABLE"  # for speed
```

Figure 1: Customization for Adaptec.

```
hostname="your.domain.com"
network_interfaces="fxp0 lo0"
ifconfig_fxp0="inet 123.456.789.012 netmask 255.255.255.0"
nfs_client=YES
```

Figure 2: FreeBSD sysconfig file changes.

- Booteasy boot manager – 512 bytes at first drive sector
- Partition table
- FreeBSD – 200 MB partition
- C: drive – 400 MB physical partition
- D: drive – 500 MB logical partition

The method used on the 35 NLS systems was to completely build and test a single client system per the above. Then this system was booted onto the network with a DOS boot floppy, and CSCDUPE was used to backup the FreeBSD image across the network to our server running Samba.[14] This uncompressed 200 MB image is then downloaded onto the other 34 client systems in the same manner; the `sysconfig` and `hosts` files are configured on each system.

The CSCDUPE backup and multiple restore is done from the first sector on the drive, through the first sector of the C: partition. This means that Booteasy, the partition table, and the FreeBSD partition are all in the image. The CSCDUPE restore requires 45 minutes for this 200 MB image, but several can be started at once. All that remains to be done on each client system after a CSCDUPE image restore, is to assign D: as a logical drive with DOS FDISK (since the D: partition is not copied), and format the C: and D: partitions as FAT. The individual client systems are now ready to be turned over to the test community for installing operating systems.

Server System Description and Configuration

The server platform is based on an Intel Venus Pro-200 motherboard,[15] with 128 MB of RAM, Adaptec 2940/3940, and two Intel 100 MBit ethernet cards. This FreeBSD server supports 90+ client systems.

The disk space mounted under `/array` on the server is composed of eight 9 GB SCSI drives (72 GB), which formats to 66 GB. To keep from having to mount and keep track of these drives, the Concatenated Disk Driver support is used in FreeBSD. Below is the line that need to be added to the kernel configuration file:

```
pseudo-device ccd 4
        # ccd driver support
```

In addition, the following changes need to be made in `sysconfig`, in addition to standard customization changes:

```
nfs_client=YES
nfs_server=YES
gateway=YES
```

Our eight 9 GB drives are configured in `ccd.conf` (yours may be different):

```
ccd0 32 none /dev/sd8e /dev/sd12e
    /dev/sd9e /dev/sd13e /dev/sd10e
    /dev/sd14e /dev/sd11e /dev/sd15e
```

We mount this ccd array in `fstab`:

```
/dev/ccd0c /array ufs rw 1 2
```

Verify the below lines are still in /etc/rc to startup the disk array:

```
# Configure ccd devices:
if [ -f /etc/ccd.conf ]
    ccdconfig -C
```

All that remains is to keep the image server's filesystem backed up via DLT tape to preserve the `/array` directory structure, the created disk images, and associated scripts. A drawback with CCD is that any drive that fails ruins the entire data array, and this has already happened once. However, a good feature of CCD is data striping across multiple disk drives, which raises the throughput over that achieved by a single drive. Our CCD array of eight SCSI-2 drives Bonnie benchmarks[16] at 9.5 MB/sec, which is about equivalent to the theoretical throughput of a single 100 Mbit ethernet interface.

The server contains two Intel EtherExpress Pro/100 TX PCI ethernet cards,[17] one which interfaces to the private class-C network of 35 NLS "PC Hardware All Alike," and the other which interfaces to the rest of the client systems covered by the "PC Hardware All Different" Web page. The server also is an IP gateway, which allows the 35 NLS systems to obtain access to the rest of the network, while isolating them from IP, Novell, Netbios, and Appletalk traffic on the rest of the network.

The server runs a DHCP daemon, which is not part of the FreeBSD distribution.[18] This daemon listens for DHCP requests from the NLS client systems, and assigns them a unique IP address on the private network each time they are booted into Windows or OS/2. The DHCP daemon assigns each client system the same IP address configured in the FreeBSD partition on that system.

Users setup each OS to obtain the IP configuration via DHCP, which keeps down configuration errors, especially over a large number of possible images. This also allows multiple copies of an OS/language to be running on different client systems at the same time for any special debugging efforts.

The Apache web server is running on this server system, and it supplies the two different Web pages to users. Apache installation or configuration will not be covered here.

[14] http://lake.canberra.edu.au/pub/samba/.
[15] http://developer.intel.com/design/motherbd/vs/index.htm.
[16] http://www.freebsd.org/ports/benchmarks.html.
[17] http://www.intel.com/network/doc/6285/index.htm.
[18] http://www.isc.org/isc/dhcp.html.

The server also runs Samba, which supplies the protocol used to duplicate each client system with CSCDUPE, and this protocol is used during the Language Workshops for file transfers and loading new code releases.

Backup Process Description

Once the user initiates FreeBSD with "F1," FreeBSD boots normally and NFS mounts the host server filesystem, which we have defined on each client system as `/array`. Just before FreeBSD exits `rc.local`, it runs a Perl[19] script from the NFS mounted directory called `/array/bin/task`, which does the following:

- Obtains the `$hostname` of the client system, and the `$date` (for other uses later).
- Opens (for append) a log file on `/array/log` that is used for statistical purposes.
- Re-makes the hard drive device id, in case the user changed the C: partition size.
- Checks for the presence of some information files about each client system, and if not present, `task` runs and records the output of commands `dmesg`, `fdisk`, and `disklabel`. These three files are stored into: `/array/machines/$hostname/info/`, and are backed up when the entire `/array` structure is backed up. In case of disaster, all the setup information about a client system can be obtained from them.
- Runs the `ntpdate` command to set the time on the system, in case it has drifted.
- Finally, `task` checks for the presence of a file: `/array/machines/$hostname/backup`, which contains the name of the desired image to be backed up, i.e., `uk_win.dd.gz`. This file was created by user interaction with the Web interface, and will be explained later.
- The `task` script then prints a line to the client system console, letting the user know that the backup has started, and issues the following command:

```
'dd if=/dev/sd0s2c | gzip -1 > \
/array/machines/$hostname/\
images/$image';
```

- The `dd` utility reads the data from the client system hard drive (the C: partition is specified as `sd0s2c` above).
- The `dd` data is compressed by `gzip` on the client system, using the fastest rate option.[20]
- The image file is transferred across the network, and stored on the server disk array at: `/array/machines/hostname/images/$image`, where `$image` is a unique filename that that associates it with the OS and language.
- After the backup is complete, throughput numbers are displayed and recorded in the log file, which is then closed.
- The stored image on the server now has a unique filename that associates it with the OS and language.
- Finally, the client system is shutdown.

Compression efficiency of a user disk image varies with the OS and support applications installed on the 400 MB C: partition. Our average compression ratio, based on 63 NLS images (Win31, Win95, WinNT, and OS/2) is almost exactly 4:1.

To improve compression of the client system image, zeros are stored in all unused areas of the C: partition. Gzip'ed zeros consume very little space in the stored server image. The Norton WIPEINFO program is used on the C: drive after it has been initially formatted, but before the user begins the construction of an OS on the drive.[21] During a trial, we were able to store a 400 MB disk image of all zeros in about 150 KB.

Restore Process Description

The user schedules a restore on the Web page, and is reminded that the proper language keyboard needs to be plugged into the client system. Once the user boots into FreeBSD with "F1," FreeBSD NFS mounts `/array`. All of the previous steps described above for a backup are done by `/array/bin/task`, with these differences:

- The `task` script prints a line to the client system console, letting the user know that the restore has started, and issues the following command:

```
'gunzip < /array/machines/\
$hostname/images/$image | \
dd bs=27136 of=/dev/sd0s2c';
```

- The desired disk image is read from the server across the network and is uncompressed by `gunzip` on the client system.
- The `dd` utility takes the output pipe from `gunzip`, and writes the data to the client system hard drive (the C: partition is specified above).
- After the restore is complete, throughput numbers are displayed on the client system and recorded in the log file, which is then closed.
- Finally, the client system is shutdown and rebooted.

[19] Redundant footnote. If names Wall, Christiansen, or Schwartz don't ring a bell, seek professional help.

[20] We settled on the "-1" option, the fastest compression method. See "man gzip."

[21] http:/www.symantec.com/nu/fs_nu8win.html. WIPEINFO.EXE is part of Norton Utilities 8.0.

NLS Web Page Description – "PC Hardware All Alike"

This Web page is named `nlslab2.html`. This page has pulldown menus to select each client system, the Languages, and the Operating System desired. It includes a radio button to select either Backup or Restore, and demands a password for Backups.

The web page fields can be edited to add/remove Machines, Languages, and Operating Systems. Combinations of these fields become the unique filenames used for the images.

CGI scripts called from web page buttons perform other useful functions:

- `nlsaction.cgi` – Main cgi script for the page.
- `nlsview.cgi` – View the scheduled backup and restores in process.
- `nlsimages.cgi` – View the available images stored, and their size/dates.
- `nlsstats.cgi` – View backup and restore statistics.

NLS Lab Performance Observations

The bottleneck seems to be how fast the Dell client system can read and write the hard drive. However, the compressed partition traffic may still consume considerable network bandwidth, and plugging multiple client systems into a 10 Mbit ethernet hub can cause an unacceptable collision rate. Initial trials with ethernet hubs proved unsatisfactory when more than one backup or restore operation was done simultaneously. Presently, an ethernet switch is used which has a 100 Mbit uplink port for server data, and in theory should be able to handle ten client systems at 10 Mbit. Each client system occupies it's own 10 Mbit switched ethernet port, while several network printers are on a hub attached to the 10 Mbit switch. More testing and verification is being done in this area to find the best infrastructure performance.

The Adaptec SCB paging in the FreeBSD kernel for tag-queuing disk drives speeds disk transfers significantly.[22] An incident occurred that slowed our restore process by a factor of four, and SCB paging was found accidentally turned off.[23] Slower SCSI or IDE drives can be used, with lower performance expectations.

An interesting aspect of this method is that the majority of the processor-intensive work (compression and uncompression of images) is distributed among the client systems, and the file server is only lightly loaded serving NFS to them.

[22]Refer to FreeBSD SCB paging description by Justin T. Gibbs in /usr/src/sys/i386/scsi/aic7xxx.c.

[23]NT 4.0 turns off this Adaptec feature when it loads.

Web Page Description – "PC Hardware All Different"

Separate from the NLS infrastructure realm, this newer Web page is popular with the rest of the Test and Development users that want this backup/restore capability, but who are not in the foreign language testing business. This method uses many of the same concepts as the NLS web page for "PC Hardware All Alike."

This web page is really a redirected CGI script called `index.cgi`. This method is different from the one for "PC Hardware All Alike," in that it searches a directory called `/array/people` to find all users, and then sorts and lists the client systems assigned to them.

Each client system is then listed on a separate line by hostname, with pulldowns to select Backup/Restore, a password field, a text-entry field to enter a new image name, and a Submit button. The CGI script behind this page is `action.cgi`, and processes the requests from this page.

Groups of client systems may be assigned to a user. Each user has an assigned password, and each system has a password, either of which will allow the scheduling of a backup or restore. Because a larger number of people are associated with this web page, demanding a password for both backups and restores reduces the possibilities for user error.

The hostname on each line of the web page is also a link to a maintenance page for each client system, and is driven by the CGI script `machine.cgi`. This script provides information about the owner, and statistics about the backups/restores performed on that system. Through another script named `maint.cgi`, the owner is allowed to delete an existing image for that client system, or cancel a scheduled backup/restore that has not started.

CGI scripts hidden behind radio buttons perform other functions:

- `viewlab.cgi` – View the scheduled backup/restores in process.
- `statlab.cgi` – View the overall backup and restore statistics.

"PC Hardware All Different" Client System Duplication

The method to install these client systems uses a FreeBSD boot diskette to perform an FTP installation, and then the system is manually configured. FTP installation is used because all client systems have ethernet, and we can get the latest FreeBSD release from our FTP server. FreeBSD supports a number of popular ethernet cards, and good results have been obtained with Intel Pro/100B's and 3Com chipsets, at both 10 and 100 Mbit rates.

The minimum SCSI drive size has been 1 GB, which leaves 800 MB for the C: partition.[24] This remaining space can be used any way the user wants, and filesystem types can be any format, such as FAT, NTFS, HPFS, etc.

Usage Statistics, System Cost, and ROI

Usage statistics are available from both web pages. When the statistics button is pressed, a perl script parses through the log files kept on the server, and presents them to the requester. The statistics show the users getting the most benefit from the system, and perhaps which users need to drop off this strategy. These scripts are:

- `nlsstats.cgi` – NLS "PC Hardware All Alike" statistics
- `statlab.cgi` – "PC Hardware All Different" statistics

The 35 Dell client systems described here cost over $100K. The server to support them was pieced together for about $20K. This backup and restore method may not suit everyone. This is an on-demand system, and users expect restores within minutes of scheduling them. Using tape, CD-ROM, boot diskettes, or other slower means of image restore would not be acceptable in our case. We decline to bring aboard users that just want to hold the contents of their office PC in case it crashes. If they actively use it to test software however, this method is usually their best choice.

Security

Our biggest problem is when a user is inattentive to the pulldown selections. The worst case scenario is when a user overwrites an image on the server, due to an error specifying a backup. Passwords are needed to backup an image on the NLS page, but are not necessary for restores. However, passwords are required for any operation on the "PC Hardware All Different" web page.

Support Software

This paper, all the scripts described within, and any other documentation necessary to duplicate this backup/restore system, will be available for anonymous FTP from `heathers.stdio.com/pub/lisa97/`.

Author Information

Tyler Barnett joined IBM in 1977 as a Field Engineer in Louisville, KY. He left the University of Kentucky in 1989 with an BSEE. He is now in the software test organization at Lexmark International, where he is responsible for sysadmin, printer performance, and whitebox testing. His mail address is: Lexmark International, B035-3-2D4, 740 New Circle Road, Lexington, KY 40511. Reach him by email at <tbarnett@lexmark.com> or <n4ty@stdio.com>.

[24] We use 1 GB Seagate ST31055W, and 2 GB Seagate ST32171W. Both discontinued and inexpensive.

Managing PC Operating Systems with a Revision Control System

Gottfried Rudorfer – Vienna University of Economics and Business Administration

ABSTRACT

During the lifetime of a workstation the system administrator is faced with constant change in system configuration (updates, new software). The users of a workstation, too, may change the system configuration. We describe the necessary concepts for maintaining many similar configured PC clients in a lab environment. The main software component consists of *fsrcs*, a revision control system similar to *RCS* and *SCCS*. The revision control software is available without charge.

Motivation

This paper was motivated by the need to reduce the huge administrative effort for running a computer training room at the Department of Applied Computer Science. It is the first attempt at our university of a fully automatic client installation and update solution using free software. The students can handle the following from the boot-prompt of the PC client without assistance of the system administrator:

1. Repair or install Linux software (operating system and applications) on the PC.
2. Repair or install Microsoft Windows 95 software (operating system and applications) on the PC.
3. The installation can be requested by the user at any time.
4. The user can decide to update both operating systems or just Windows 95 or just Linux.

Administration tasks are reduced because the system administrator installs new software only on one PC. After installation a set of programs help the system administrator to define a new master copy for all clients on the server.

Comparison with other tools

There are many tools available for managing software environments [CW92, Fut95, Har94, Jam94, PS94, Rid94, VCV92]. A new draft standard for software administration tries to define interfaces and formats for the administration of a network of heterogeneous systems [Arc93]. Many tools use software packages and supply commands for the administration of these packages. Microsoft's Systems Management Server (SMS) distributes software packages (i.e., Microsoft Office) and runs unattended installation. Limited functionality is provided for automatic checks of an existing package on the PC. Installation scripts have to be written for local customizations, patches, and not SMS aware software. This is a very complex task.

Other approaches just extract a ZIP- or a compressed tar-archive. This approach is not well suited for checking the correct installation of the software. Files on the PC which are not part of the software installation might remain after the extraction of the archive. Unfortunately, many of the above tools only provide a partial solution for our software distribution problem. We often have a situation where the software is already installed, but then a user modifies some files of the installation. We need a system that checks and repairs an already existing installation as fast as possible. Users may not accept the system if this process is too time-consuming.

Our approach is based on the fact, that the same software packages installed on PCs result in a very similar set of files even if the PCs have a different hardware configuration. Compared to the above approaches, our approach tries to replicate already installed software to other PCs. The necessary configuration of the software is done afterwards. Our approach just distributes files without looking at the semantics, if possible. However, we have to look into files if they contain parameters to configure for proper operation of the software. The operating system Linux has many tools and concepts for software management of PCs available [Tro96]. Therefore, we decided to develop our programs under Linux.

Implementation

The management software consists of three parts (see Figure 1):

- a set of programs for downloads from the server to the clients
- a set of programs for uploads from the clients to the server
- a revision control system for file trees

Implementation of the download programs

On the PC Client Side

The root file system of the installation program is loaded into the main memory to avoid any conflict with the hard disk. This is done by loading the file system as initial ram disk [AL96]. If the PC cannot boot from a prior installation, the same kernel and the initial ram disk are loaded from a floppy disk.

The normal usage is to load the kernel and the ram disk from the Linux file system of the already installed PC by entering the boot option "c" (update both Linux and Windows 95) or "d" (update Windows 95 only) or "e" (update Linux only) at the LILO [Alm96] prompt.

When the initial ram disk is mounted the script */linuxrc* is executed. The script configures the network interface with a *bootp* request. On success the NFS exported directories of the server are mounted read-only and other libraries and programs are made available.

Finally, the installing perl script is executed. This script does the following:

1. Get the system time from the server and write it to the CMOS clock.
2. Check the partition table and if it does not correspond to the sample table recreate the partition(s).
3. Repair the file systems if they are inconsistent.
4. Mount the file systems and enable swap.
5. Ask the server to update the client.
6. Detach the mounted file systems.
7. Write a new master boot record.
8. Do local customizations (i.e., add the hostname to the registry of Microsoft Windows 95)

There is no reboot necessary during installation.

On the Server Side

The regular login shell of the download user *pc7inst* is replaced by the program *toclient* which first changes the effective root directory (*chroot*) to home of the master copy. The Set-user-ID permissions bit of the program is set to run the program as user *root*. All *root* users of the clients have access to this account via the *.rhosts* file. The name of the client and the mode of update (all operating systems or just Linux or Windows 95) are passed as command line arguments to the program *toclient*. Finally the program *rdist* [Coo92] is executed.

Implementation of the Upload Programs

The implementation of the upload programs tries to preserve the security on the server with an upload user *pc7adm*. This user has *root* privileges on the Linux operating system for the installation on the clients (is added to the *.rhosts* file). The core of the upload program is *rdist*, too. The regular login shell of the upload user *pc7adm* is replaced by the program *ask*:

1. If the system administrator logs in successfully at the server as *pc7adm* the program is called without arguments and asks the user which client to upload. When the client is found in the database, access of the *root* user from the client is granted. Finally the program starts a remote shell on the client with a slightly modified *rdist*.
2. The client program *rdist* connects via *rsh* to the upload user *pc7adm* and executes the Set-user-ID root program *toserver -S*. This program first changes the effective root directory to the root of the client master installation. Then the *rdist* server *rdistd -S* (which is statically linked) is executed. The files are installed with the *rdist* options *-onumchkgroup -onumchkowner* to use the numeric group and user ID for checking

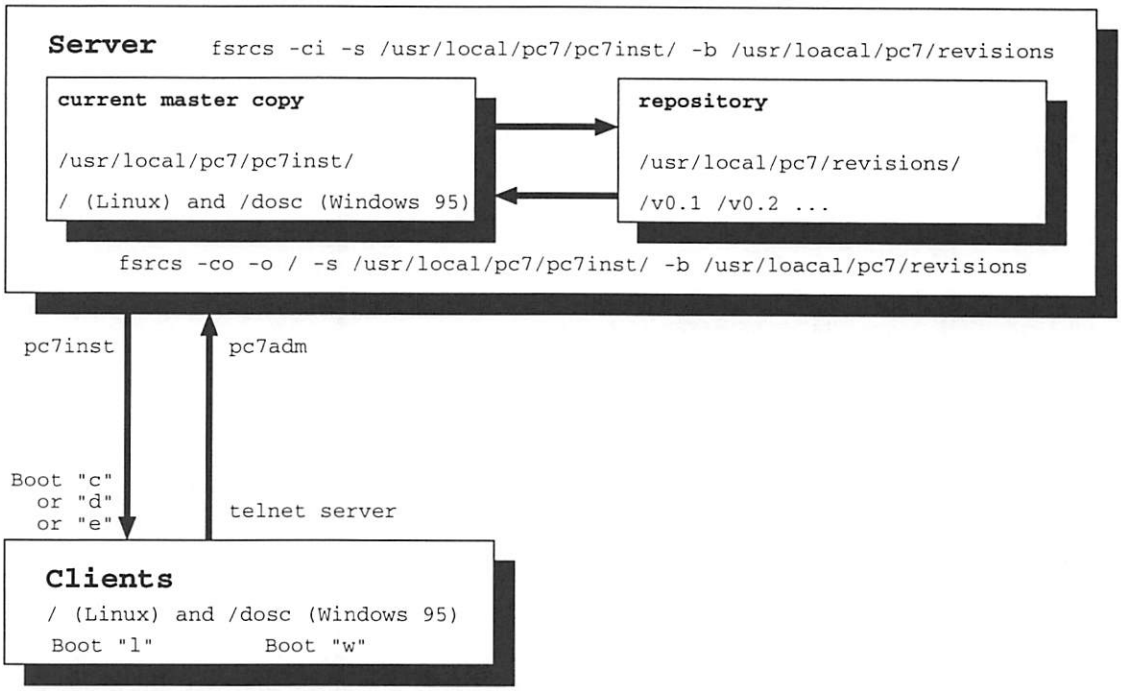

Figure 1: Relationship between clients, server and the repository.

group and user ownership because the group and user name might not exist on the server.

3. Finally, the access of the client is removed from the *.rhosts* file.

Implementation of the Revision Control System

The first version of our management software had no revision control system. After some testing we found that one version of the master installation has the disadvantage of no ability to downgrade after an update with errors.

We decided to store the master copy in the repository of our file system revision control system *fsrcs*. It can handle at least text and binary files, symbolic and hard links, character and block device files and directories with proper access permissions and ownership.

The current implementation stores the master copy on the server. A new version of the master copy is manually created when a major change in the software configuration of the clients occurred.

Multiple revisions can be managed by our system. The repository is implemented as a file-tree. Underneath the root there are directories with the name of each revision. New or changed files (and directories and links) are stored inside each version-directory. However, a symbolic link to the original file in the repository is created when there was no change. Symbolic and hard links are stored in separate directories for each version. The initial repository is created by copying the whole tree into the directory of the first version.

The program is implemented in the programming language *perl* [WCSP96].

Check-In

This section describes the updating process, done on the check-in of the new version. We distinguish a different behavior for files (plain files, character and block special files), symbolic links, hard links and directories. The procedure for directories is fairly obvious. If a directory is found, the check-in function is called with its name (recursion). Directories are created with the access rights and ownership of the original. If a directory does no longer exist in the new version, no reference to the previous version is made. Files are compared with the current version of the file in the repository if it exists. If the size, permissions, ownership and group are the same, this file is first marked as unchanged. The check-in function performs link optimization.

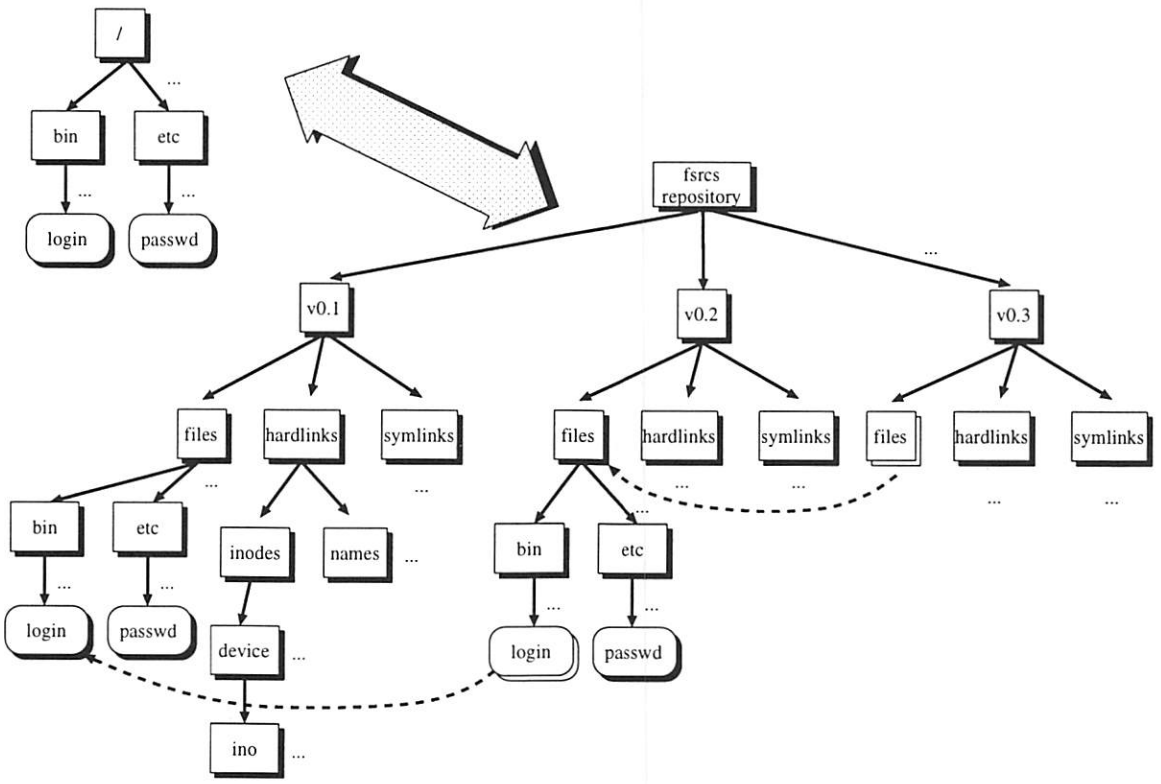

Figure 2: Internal representation of file system objects. At check-in time of version v0.2 the file `/bin/login` has not been modified and the file `/etc/passwd` has been changed compared to the first version. The check-in function made a reference to the previous version of `/bin/login`. All files of version v0.3 have not been changed at check-in time compared to the previous version v0.2. The check-in function made a reference at the highest possible level to the prior version with link optimization.

If a whole subtree is unchanged, a symbolic link is created at the highest possible level in the repository. If a reference to a prior version of an object within the repository is necessary, another link optimization is done. The check-in function does not simply create a symbolic link to the prior version, instead it creates a reference to the original object in the repository. These two levels of link optimization guarantee a minimum number of symbolic links in the repository. Compared with *RCS* and *SCCS* [BB95], the smallest piece is an entry of a directory. Your software does not bother about the differences between text or binary files.

Check-Out

On check-out, the program compares the entries of the given check-out directory with the entries in the repository. Missing entries are created and entries in the checkout-directory that are not in the repository are recursively removed (directories) or unlinked.

Limitations

The current version of *fsrcs* has no support for access control lists (ACL). Some UNIX operating systems support ACL which allows the file owner to permit or deny access to a list of users. However, this feature is neither used nor supported by Linux and Windows 95 installations. Another restriction exists for symbolic links. Many UNIX operating systems do not set the permissions for symbolic links correctly. On repeated checkouts symbolic links will probably be recreated even if they have not been changed.

Availability

Our file system revision control system is available from `ftp://ftpai.wu-wien.ac.at/pub/fsrcs/fsrcs.tar.gz`.

Administration tasks

The time to administer a set of PCs is reduced to the installation or update of new software on a single PC. Our tools guarantee the correct installation of the software on all other PCs. If some files or partitions have been changed, our tool repairs the installation automatically and quickly without user interaction.

Installing New Software on a PC

Administration sessions require the following steps to install or upgrade software:

1. Make a full update (boot with option "c").
2. Uninstall the prior version of the software, if required.
3. Install the new version of the software.
4. Upload the new installation to the server.

The new software is transferred to other clients when they make a full update (boot with option "c").

Finding the Correct Configuration

The most complicated task is searching for mandatory customizations between different PCs. First, we tried to eliminate individual configuration parameters, if possible.

Linux

Linux software distributions often configure their network parameters statically. A networked Linux PC needs a unique IP address and hostname. First we installed the Linux distribution Redhat 4.0 (Colgate) and then replaced the file */etc/rc.d/init.d/network* with the file */etc/rc.d/init.d/rc.bootp* of the package *bootpc* [Haw96]. This package demands the network parameters with a *bootp* request from the server, configures the network interface and creates the files */etc/hosts* and */etc/resolv.conf*. After this change in the configuration of Linux, there are no individual configuration parameters left.

Windows 95

We configured IP networking to obtain the network parameters with a *DHCP* request from the server. Windows 95 uses the binary files *system.dat* and *user.dat* to store its configuration. These files are modified by the program *regedit* (see Table 1) of Windows 95 which is called from *dosemu* [LS97] after the file checking phase. Plug and play (PNP) cards need entries in the registry, too. Otherwise Windows 95 will wrongly detect new hardware and try to install new software. First we exported the registry as text files of two different PCs using *regedit*. Then we looked at the differences between the files with the program *diff*. We found that the necessary entries for our PNP sound cards are in `HKEY_LOCAL_MACHINE\Enum\ISAPNP\`. We generated a similar text file in which we replaced a generic string with the PNP-ID of the sound card.

```
REGEDIT4

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\VNETSUP]
"ComputerName"="hostname"
"Workgroup"="PC7"

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\control\ComputerName\ComputerName]
"ComputerName"="hostname"
```

Table 1: The contents of a configuration file for the registry of Windows 95. First, the string hostname is replaced by the actual hostname of the PC. Finally *regedit* is called with the modified file to patch the registry.

Periodical Checks

The cron daemon of each PC is configured to run our management software each day early in the morning.

Performance

For software distribution a UNIX server with Pentium processor, approx. 128 MByte main memory and a PCI based network interface will be enough to serve approx. 30 client PCs.

Our hardware consists of one Digital Alpha Server 4000 5/300 (2 × 300 MHz CPUs, 1 GByte main memory, 20 GByte hard disk, 2 × 100 Mbps full duplex ethernet cards), one Cisco Catalyst switching hub, 27 PCs (Pentium Pro 200 MHz CPU, 64 MByte main memory, 2 GByte hard disk, 10/100 Mbps ethernet card currently used in 10 Mbps mode).

The size of Windows 95 and applications is currently 202 MBytes (3413 files in 194 directories). The size of Linux is currently 707 MByte (38372 files, 2797 symbolic links, 735 character special files, 304 block special files, 706 hard links in 2171 directories)

Installation from scratch is done with the prior described kernel and root file system on a boot floppy which requires approx. 42 minutes for partitioning, formatting, checking and copying of both operating systems. After automatic reboot of the installation system, the two operating systems are available.

Checking of the whole installation (Linux and Microsoft Windows 95) with minor modifications requires approx. eight min. Checking our installation of Microsoft Windows 95 requires approx. two min.

Each of the 27 clients needs about 20 minutes when all of them check their whole installation at the same time against the server. In this case the server is the limiting resource. Currently, the server is not tuned for optimal performance.

Conclusion

By using open operating systems for systems management, it was easy to develop a powerful tool for UNIX and Windows 95 operating systems. The users are greatly satisfied with the ability to repair the local installation by themselves if something goes wrong with the software. The time spent on troubleshooting has decreased by 90%.

Future Work

This implementation proves, that Linux is a suitable operating system for automated installation and update of software.

This version of software requires PCs with an equal hardware configuration. We plan to extend the shown concept to handle different hardware installations by extending the revision control system to find the differences between two software installations. The program *rdist* seems to be the limiting factor on the server side. There is one independent *rdist* process for each client which scans the state of each file. This causes a heavy system load on the server when many clients perform an update at the same time. This problem could be solved by extending *rdist* using a state database. Another area of improvement is in adding revision control capabilities to *rdist* directly.

Author Information

Gottfried Rudorfer works at the Department of Applied Computer Science at the University of Economics and Business Administration where he received his MBA. His research interests are centered on issues of distributed database systems, administration of heterogeneous computing environments, operations research and artificial intelligence. He has been a system administrator since 1993. Reach him via mail at University of Economics, Augasse 2-6, A-1090 Vienna, Austria; or electronically at Gottfried.Rudorfer@wu-wien.ac.at.

Bibliography

[Al96] Werner Almesberger and Hans Lermen, "Using the initial RAM disk (initrd)," `ftp://ftp.funet.fi/pub/Linux/PEOPLE/Linus/v2.0/linux-2.0.30.tar.gz`, almesber@lrc.epfl.ch, lermen@elserv.ffm.fgan.de, 1996.

[Alm96] Werner Almesberger "Generic boot loader for Linux," `ftp://lrcftp.epfl.ch/pub/linux/local/lilo/lilo.19.tar.gz`, almesber@lrc.epfl.ch, 1996.

[Arc93] Barrie Archer, "Towards a POSIX Standard for Software Administration," *LISA,* pp. 67-79, November 1-5, 1993.

[BB95] Don Bolinger and Tan Bronson, *Applying RCS and SCCS*, O'Reilly & Associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 95472, 1995.

[Coo92] Michael A. Cooper, "Overhauling Rdist for the '90s," *LISA VI*, pp. 175-182, October 19-23, 1992.

[CW92] Wallace Colyer and Walter Wong, "Depot: A Tool for Managing Software Environments," *LISA VI*, pp. 153-162, October 19-23, 1992.

[Fut95] Atusushi Futakata, "Patch Control Mechanism for Large Scale Software," *LISA IX*, pp. 213-219, September 17-22, 1995.

[Har94] Magnus Harlander, "Central System Administration in a Heterogeneous Unix Environment: GeNUAdmin," *LISA*, pp. 1-8, September 19-23, 1994.

[Haw96] Charles Hawkins, "Linux Bootp Client," `http://www.damtp.cam.ac.uk/linux/bootpc/`, ceh@eng.cam.ac.uk, 1996.

[Jam94] Kevin Jameson, "Multi-Platform Code Management," O'Reilly Associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 95472, 1994.

[LS97] Matthias Lautner and Robert Sanders, "DOSEMU PC Emulator," `ftp://tsx-11.mit.edu/pub/linux/ALPHA/dosemu/`, linux-msdos@vger.rutgers.edu, 1997.

[PS94] Dieter Pukatzki and Johann Schuhmann, "AUTOLOAD: The Network Management System," *LISA*, pp. 9-17, September 19-23, 1994,

[Rid94] Paul Riddle, "Automated Upgrades in a Lab Environment," *LISA,* pp. 33-36, September 19-23, 1994.

[Tro96] Jim Trocki, "PC Administration Tools: Using Linux to Manage Personal Computers," *LISA X,* September 29-October 4, 1996.

[VCV92] Ram R. Vangala, Michael J. Cripps, Raj G. Varadarajan, "Software Distribution and Management in a Networked Environment," *LISA VI,* pp. 163-170, October 19-23, 1992,

[WCSP96] Larry Wall, Tom Christiansen, Randal L. Schwartz, and Stephan Potter, *Programming Perl,* Second Edition, O'Reilly Associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 95472, 1996.

Bal – A Tool to Synchronize Document Collections Between Computers

Jürgen Christoffel – GMD

ABSTRACT

The enormous growth of computer networks and declining hardware prices allow people to have more than one computer, e.g., to use a primary computer at work, a laptop or notebook as a secondary computer for 'on the road' and maybe a third computer at home.

One non-trivial problem that users face when using more than one computer alternately is the difficulty of keeping multiple, distributed copies of their files up to date and in sync, if they can't permanently share resources between their machines, e.g., when using a notebook computer off-line for some time.

System administrators of Unix sites who need to automatically distribute files between machines have known such problems for years and have come up with various tools to solve their specific needs. But tools which help Unix administrators are not always adequate for their users too.

This paper describes *Bal*, a tool which enables Unix users to more easily keep distributed copies of their files in sync. *Bal* is written in Perl and keeps two directory trees in sync.

Introduction

Document collections are sets of documents which are in some way related. For the purpose of this paper a document collection is a set of files or directories or both which are handled by Bal as a whole.

Users who alternately use multiple computers to do their work will sooner or later face the problem of keeping their documents in sync. As long as they use computers which are connected to the same local-area network it is easy to use a client-server approach to keep files on one machine and share them via some means like NFS. But as soon as one of the computers is not always connected to the network the problems will arise because users keep distributed copies of their documents and have to synchronize those document collections between machines.

Maintaining distributed copies of such collections can be a nightmare if users can't always guarantee that they will only change one of their document collections at once before merging changes between their computers.

The Macintosh and PC user communities have for some years had the benefit of programs like PowerMerge [LEADER] or Laplink [LAPLINK] or the "Briefcases" built into Windows 95 and NT [BRIEFCASE] which keep two document collections in sync. Unix users until now have had to use whatever tools that were available to at least automate some of the work to be done to keep their directories in sync.

Bal – named for the balancing effect it has on distributed document collections – helps users and system administrators keep track of changes made to files or directories and telling them when files have changed, have been deleted, or new files have been created on either side. It can automatically copy, create, or delete files to synchronize two document collections. It warns users if files have been changed in both collections since its last run and offers help in resolving the conflicts.

Comparisons

Modern distributed file systems [TANEN] like the Andrew File System [AFS] address some of the problems but are not always easily available.

While version control software like *rcs*(1) [RCS] or *cvs*(1) [CVS] could be used in theory, these tools are inadequate for keeping home directories in sync in practice because of the overhead and restrictions imposed by them. Bal is intended to manage typical user files like mailboxes or calendar files and version control systems where not designed to handle these. Version control tools regularly keep a third copy of each document on disk and they have special commands for checking documents in or out which need to be remembered.

Tools like *rdist*(1) [RDIST] or *track*(1) [TRACK] which are adequate for specific jobs in the system administrator area don't help users (and system administrators) much because such tools have been designed for an asymmetric problem, namely to distribute master copies of files between servers and clients, which allows overwriting of files on the client.

Automatic file distribution tools like rdist or track copy files between computers using either a *push* model where a master server updates some clients periodically or a *pull* model where each client periodi-

cally queries a master server to retrieve the freshest files.

Both models are very helpful for automated software distribution where, e.g., system administrators need to periodically update system files on a cluster of machines. Tools like rdist and track are thus in widespread use on Unix machines.

The scheme used by such tools is *asymmetric* since it allows one master copy of a set of files on a server which is distributed to possibly multiple clients which are not expected to make changes to their copies. This scheme is not appropriate for users who want to keep distributed document collections in sync.

Users face a *symmetric* problem instead, because both document collections resemble different versions without a master copy or central repository. These document collections need to be merged or synchronized, i.e., documents may have to move both ways and checks have to be made to ensure that changes made in either document collection aren't lost.

Because Bal is intended to be used interactively – it needs user interaction to decide how to resolve potential conflicts – it doesn't face the security problems of the rdist family of tools.

Implementation Issues

Bal is a program to maintain identical copies of document collections on different hosts. It preserves the owner, group, mode, and modification time of files if possible.

Bal is written in Perl [PERL] and uses the high-level constructs like associative arrays provided by Perl to manage its internal data. It uses Perl's interfaces to the various DBM implementations for Unix [HASH] to persistently store information about the state of each document collection between sessions. Which DBM implementation is used is configurable.

Like rdist and similar tools, Bal uses a configuration file which specifies the files of the document collections to keep track of and what actions to take when synchronizing the document collections. The configuration file specifies the left and right document collections respectively in the notation used by rsh/rcp. It also allows to specify via Perl's regular expressions which files to exclude from synchronization.

To properly synchronize two document collections Bal needs to keep track of the state of each document in either collection in order to decide what to do to synchronize them. When first run it compares the two document collections and creates a reference database which contains state information for each file in both collections.

When Bal compares files for the first time, it compares the attributes of each pair of files. Attributes of interest are the *file name, device number, inode number, permissions, number of hard links, uid, gid, size* and the *modification time* of each file. For each pair of files it uses the modification dates to see which one is oldest. All checked attributes will be stored in the database for reference when Bal is run again.

On subsequent runs Bal checks the document collections against the reference database to see what has changed. After synchronizing the two document collections it stores the new state in its database. During synchronization it may encounter one of the following situations:

- a file hasn't changed in either of the document collections
- a file has changed (i.e., deleted, edited or renamed) in one document collection and needs to be updated in the other collection
- a file has been changed (i.e., edited or renamed) in both document collections

The last situation is obviously the most problematic one and Bal doesn't solve them automatically. Instead it copies conflicting files into special subdirectories and suggests various checks to help the user resolve the conflict, among them using *diff*(1) or *cmp*(1) to see if files have only been *touch*(1)ed.

When Bal is run it gathers the relevant file attributes from both the left and the right document collection and compares the attributes to those stored in the reference database. The reference database allows Bal to detect changes to the same file on both sides by comparing the file's attributes on disk with the attributes stored in the reference database.

To actually change files Bal depends on rsh/ssh and rcp/scp to transfer documents between machines and it generates the appropriate commands which it then executes.

Bal can be configured to run in batch mode where it will send a report of the synchronization process via email.

Example Session with Bal

During initialization files are normally considered different if their modification time and size disagree. But when the document collections are not in sync during initialization of the reference database, Bal can be told to use a checksum algorithm like md5 to compare pairs of files and check whether files with different timestamps but same size are actually the same. Using a checksum like md5 additionally allows to find duplicates which have different names.

Changed Files

Once the reference database exists, later runs only need to compare the attributes of each file with those stored in the database. When Bal encounters a file which has changed on one side, it generates the appropriate shell commands to bring both sides in sync again.

The following example shows a Bal run where *priv.bal* is the reference database where the attributes seen on the last run have been stored. The document

collections reside on hosts *aeppel* and *vortex* and Bal is running on *vortex*. The -v flag tells Bal to output the commands it executes; see Figure 1.

Bal copies the newer version to the other side and updates the attributes stored in the reference database for both files.

Conflicting Files

When Bal encounters a pair of files that have both changed on the left and right since its last run, it cannot solve that case automatically. Instead it explains the conflict and outputs suggestions for resolving the conflict; see Figure 2.

Bal puts the older version of the conflicting files into a special conflicts subdirectory – *.bc* in this example – and suggests an action the user might take to resolve the conflict manually. In the example above the suggestion is to run *diff*(1) to compare the two versions.

The conflicts subdirectory is always in the same subdirectory as the original document, so it is easy to see where the document came from.

The user should manually resolve the conflict, e.g., by merging the conflicting versions into one file and then can remove the conflicts subdirectory.

Renamed Files

Bal tries to detect when a file has been renamed on either side. To detect when a file has been renamed it stores the inode number of each file in its reference database. When a file is missing from either side it uses the inode number to find the new name of the file and if the other attributes in the reference database match that file it can rename the corresponding file on the other side to the new name.

If a file has been renamed differently on both sides the current implementation of Bal handles this by simply copying both files as if it where new versions of different files. One might argue that it should signal a conflict instead.

Performance

The current implementation of Bal needs about 3% of the size of the document collections to store attributes in its reference database. On a Linux machine with a Pentium Pro 200 processor it takes about 15 seconds real time to compare two directories containing about 12 Megabytes in 3,267 files. Those numbers do not take the time into account for the actual file transfers needed to bring the directories in sync again.

Future Work

Bal is designed for interactive use and thus is a good candidate for a window-based user interface. We plan to provide an Emacs major mode to interface with Bal and a graphical user interface based on Perl/Tk in the future.

```
vortex[788]: bal -v -f priv.bal
rcp -p lesetips aeppel:priv/lesetips
rcp -p aeppel:priv/lisa-11/register-info lisa-11/register-info
rcp -p aeppel:priv/prep.consult prep.consult
rcp -p aeppel:priv/quotes quotes
rcp -p quotes.news aeppel:priv/quotes.news
rcp -p aeppel:priv/quotes~ quotes~
rcp -p aeppel:priv/signatures signatures
rcp -p aeppel:priv/xyzzy-9707.tar.gz xyzzy-9707.tar.gz
rcp -p aeppel:priv/spooky spooky
rcp -p aeppel:priv/todo todo
```

Figure 1: Example of Bal output

```
vortex[798]: bal -v -f home.bal
rcp -p .zshenv aeppel:.zshenv
rcp -p aeppel:.emacs .emacs
rcp -p Notes/8-97 and aeppel:Notes/8-97
# conflict: mtime mismatch for .netscape/bookmarks.html
#     Sun Aug  3 14:20:04 1997  .netscape/bookmarks.html
#     Wed Jul  2 19:36:58 1997  aeppel:.netscape/bookmarks.html
mkdir .netscape/.bc
rcp -p aeppel:.netscape/bookmarks.html .netscape/.bc/bookmarks.html
rcp -p .netscape/bookmarks.html aeppel:.netscape/bookmarks.html
# action: diff .netscape/bookmarks.html .netscape/.bc/bookmarks.html
```

Figure 2: Call for manual conflict resolution.

To improve the utilization of low-bandwidth connections Bal could be augmented to transfer only the modified parts of changed files. The rsync algorithm [RSYNC] could be applied to update files over low-bandwidth high-latency bi-directional communications links but that would force Bal to use it's own transport protocol instead of simply relying on rsh/ssh to do the work.

Additionally Bal could be augmented to recognize when files have been compressed and instead of transferring the compressed file, compress the file on the other side too.

A future version of Bal will be capable of synchronizing document collections even if one of them is contained inside a tar file. This will make it possible to use Bal to synchronize document collections via tapes without unpacking whole tar files.

Conclusion

Bal is especially suited to the needs of users who use computers as peers and not as clients and server. Because Bal allows transparent compression during transfers it is well suited for users who use one Unix system at work and another one at home and connect via low bandwidth dial-in connections and want better utilization of the low bandwidth connection. The author uses it regularly to synchronize his Linux workstations at home and at his office over a dial-in 64k ISDN connection.

Availability

Bal has been in beta test at the authors site since May '97. It will be made available for anonymous ftp over the Internet from ftp.gmd.de and on the Comprehensive Perl Archive Network (CPAN) once it has left the beta test phase.

Author Information

Jürgen Christoffel studied computer science at the University of Bonn, Germany. He has been working with Unix systems since 1984 and joined GMD as a system administrator in 1988. He has been a Perl user since 1989 and Perl has saved him from C since then. In his spare time he teaches courses in Perl and Emacs. His email address is christoffel@gmd.de .

References

[AFS] Andrew File System (AFS) Homepage at Transarc, http://www.transarc.com/afs/transarc.com/public/www/Public/ProdServ/Product/AFS/ .

[BRIEFCASE] *Windows 95 Resource Kit*, Microsoft Press, 1995.

[CPAN] The Comprehensive Perl Archive Network is available online at http://www.perl.org/CPAN/ .

[CVS] Brian Berliner, "CVS II: Parallelizing Software Development," *Proc. USENIX Winter 1990 Conf.*, January 22-26, 1990, Washington, D.C., pp 341-352.

[HASH] Margo Seltzer and Ozan Yigit, "A New Hashing Unix", *Proc. Usenix 1991 Winter Conf.*, January 21-25, 1991, Dallas, TX, pp 173-184.

[LAPLINK] http://www.travelingsoftware.com/products .

[LEADER] Leader Technologies, *PowerMerge User Guide*, 1992-1994, Leader Technologies, Newport Beach, CA.

[PERL] Larry Wall, Tom Christiansen und Randal Schwartz, *Programming Perl, 2nd edition*, O'Reilly and Associates, 1996.

[RCS] Don Bolinger and Tan Bronson, *Applying RCS and SCCS: From Source Control to Project Control*, O'Reilly, 1995.

[RDIST] Michael A. Cooper, "Overhauling Rdist for the '90s," *Proc. Usenix LISA VI*, October 19-23, 1992, Long Beach, CA, pp 175-188.

[RSYNC] Andrew Tridgell and Paul Mackerras, "The rsync algorithm," Australian National University, TR-CS-96-05.

[TANEN] Andrew S. Tanenbaum, Chapter 5 "Distributed File Systems," *Distributed Operating Systems*, Prentice-Hall, 1995.

[TRACK] Bjorn Satdeva and Paul M. Moriarty, "Fdist: A Domain Based File Distribution System for a Heterogeneous Environment," *Proc. Usenix LISA V*, September 30 - October 3, 1991, San Diego, CA, pp 109-126.

Increased Server Availability and Flexibility through Failover Capability

Michael R. Barber – Michigan Technological University

ABSTRACT

As computing systems become increasingly mission-critical, a high level of service availability is essential. In order to maintain service availability, it is desirable to have the ability to migrate services from one server to another while having this change remain transparent to the client machines. Although commercial solutions exist which provide automatic failover capability, they are often costly and restrictive.

Manual failover capability is useful for providing service availability in situations where there is a hardware failure, or where a server must be taken down for extended maintenance. The discussion in this document will explore what primitives can be used to construct a failover-capable system, the issues involved in any service migration, and some of the specific details about doing migration of services such as NFS, sendmail, and World Wide Web.

Although implementation-specific examples provided assume a Sun Solaris 2.x operating environment, the use of a logical volume manager, and ethernet connectivity, other flavors of UNIX may also contain the necessary building blocks needed to build a failover-capable system.

Motivation

The systems group, of which I am a member, within the Information Technology – Distributed Computing Systems department at Michigan Technological University determined it was time to upgrade our primary campus servers. One item on the list of desired functionality for the new servers was the ability to do on-the-fly service migration. After examining a few commercial solutions, we decided none of them were appropriate for us.

Due to the nature of a few of our services, it was not an option to increase availability by simply replicating services in more than one place, and using using round-robin DNS, lbnamed [1], or commercial solutions such as Uniq Software Services UPFS [2].

After some experimentation, I found we could get most of what we wanted with a homegrown increased availability solution. To my surprise, the problem of doing manual failover was not as complex as it seemed: a two way failover-capable system can be built using two computers, one or more multi-ported disks, and three or more network interfaces on each host.

Foundation

An important concept to understand is that, in this model, a group of services is tied to a hostname. This hostname is considered "public" meaning it is what clients use to acquire a given service. The unusual detail is this public hostname is not permanently tied to a particular host, and may be thought of as a pointer to the server which is currently providing the given service set.

The two main pieces of the puzzle needed to build a failover-capable system are the ability to easily swap file system ownership between two hosts, and a network configuration which allows both hosts to impersonate each other. The ability to easily move file system ownership between two hosts is best accomplished by hardware such as multi-ported disk arrays (e.g., Sun's Sparc Storage Array) which allow more than one host to be connected to the array simultaneously. Although it is possible for two hosts to share a single SCSI chain, using a multi-ported disk array is a much cleaner solution.

The two logical volume managers Sun offers (Solstice Disk Suite and Veritas Volume Manager) provide support for multi-host disk arrays. The implementation-specific examples in this paper use the multi-host features of Veritas Volume Manager, although Solstice Disk Suite has a feature called "disksets" which can be used in a similar manner. While logical volume management software is not absolutely required in order to implement a failover configuration, it greatly simplifies the problem, and also provides features which can be used to provide a higher level of data availability, such as disk mirroring or RAID-5.

When configuring the two machines it is important they mirror each other as closely as possible. Each file system on the multi-ported disk array which may need to move from host to host must look the same to both hosts. If both hosts are not configured almost identically, problems may arise at a later date when a service migration becomes necessary.

Network Connectivity

Three network interfaces are required on each host to do two-way failover in which both machines are servers and either machine is capable of filling in for the other machine. The concepts in this paper can be scaled to N nodes, given a N-ported disk array and M+1 network interfaces on each node; where $N \geq 2$ and M is the number of service sets any one server will ever need to run concurrently. Situations where $M > N$ require extra complexity. This paper will focus on bipolar two-way failover (M=N=2).

So as not to confuse switches and other hosts on the network which maintain an arp cache that maps ethernet address to IP address, each host in a failover configuration must be able to impersonate the other host on both a ethernet and IP level. Furthermore, it is important to have a network topology that will allow this.

If your pair of machines are behind a smart switch that maintains a bridge table, the switch must allow for the possibility of an ethernet/IP pair to move from one port to another and update its bridge table accordingly. From the switch's perspective, a service migration should look as though the server was unplugged from one port and plugged it into another.

With the smart hub in use at our site, having the hub update its tables was as simple as running a program on the machine to which services were migrated, which generates traffic on its NIC (Network Interface Card, in this case the ethernet interface) so the switch would notice the change. To accomplish this, I used a simple program which does a UDP broadcast which sends the message "WAKE UP" out each interface to the "discard" port of all other machines in the same broadcast domain.

Each host in a two-way failover configuration will have one private and two public interfaces. Since each of the public interfaces on one node will have a counterpart on the other node, there will be only two unique ethernet addresses among the four public interfaces on two machines. Each private interface will have its own unique ethernet address, bringing the total on both machines to 6 network interfaces with 4 unique ethernet addresses, and at least 4 unique IP addresses.

Virtual network interfaces can often be substituted for physical interfaces; unfortunately this cannot be done for the three network interfaces required for two-way failover. Although this may change in a future version of Solaris, as of Solaris 2.6, a virtual interface must assume the same MAC address as the physical interface which it is associated to. However, virtual network interfaces may still be associated with public interfaces and used for virtual mail domains and virtual web servers.

Sun workstations obtain a default MAC address from NVRAM, instead of from the ethernet hardware [3]. Because of this, any add-on ethernet interfaces will by default assume the same ethernet address as the on-board ethernet interface. This behavior can be overridden by setting the option "local-mac-address?" to "true" from the prom monitor, which will cause each NIC to have a unique MAC address. In a failover system with three ethernet addresses, you will need to explicitly redefine two of the addresses, therefore setting this option in the prom is not mandatory.

You will need to create two MAC addresses, each of which will be assigned to one NIC on each machine. Although Solaris will let you assign nearly any ethernet address to the interfaces, you should pick an address which is IEEE 802.3 [4] conformant.

A standard 48 bit ethernet address is comprised of six colon-separated octets (8 bit groups). The first (left most) octet should have the first (left most) bit unset and the second bit set. The first bit indicates individual (0) or group (1) destination address. The second bit indicates a global (0) or local (1) address.

HOST A			
	hme0	hme1	hme2
hostname:	campus0	server0	server1
ethernet address:	8:0:20:81:df:f4	40:0:8d:db:46:1	40:0:8d:db:46:b
metric:	1	0	0
normal state:	up	up	down

HOST B			
	hme0	hme1	hme2
hostname:	campus1	server0	server1
ethernet address:	8:0:20:85:84:45	40:0:8d:db:46:1	40:0:8d:db:46:b
metric:	1	0	0
normal state:	up	down	up

Table 1: Sample interface configurations.

Setting the second bit to indicate "local" means the address was not assigned by IEEE. This guarantees it will not conflict with the addresses assigned by vendors, but there is no assurance it is not in use elsewhere [4,5].

When creating local ethernet addresses, many sites set the second bit in the first octet as per IEEE 802.3 specifications, and set the last four octets to the hexadecimal equivalent of their IP address. This scheme gives the ethernet address the form of 40:0:w:x:y:z (w, x, y, z in hexadecimal) where the machine's IP address is w.x.y.z (normally given in decimal). When used consistently, this scheme ensures no unwanted duplication of addresses.

You can use the ifconfig command to set the MAC address on a NIC, for example: "ifconfig hme1 ether 40:0:20:81:d2:c" will set the ethernet address of hme1 to 40:0:20:81:d2:c.

Table 1 is an example of how one might configure the three interfaces on each host.

The interface hme1 has the same hostname and same ethernet address associated with it on both machines. However, both interfaces will never be up at the same time. The interface hme2 is configured in a similar manner. In the above case, the clients will use the names server0 and server1 to obtain services.

From a network perspective, whichever machine has interface hme1 up is server0 and whichever machine has interface hme2 up is server1. See Figure 1 for a diagram of the system layout during non-combined operation.

As shown in Figure 2, after a service migration is completed, one host providing both service sets will have both public interfaces hme1 and hme2 up, while the other host will have both public interfaces down.

A third "private" interface is needed so the machine which has both public interfaces down may still have network connectivity. In the above case, hme0 with names campus0 and campus1 are the private interfaces of the two servers.

A "metric" may be associated with a network interface. This number which is normally zero indicates how may "hops" should be added to an interface when choosing the best route. The private interface must be configured with a higher metric, giving it a less desirable route so the public interfaces will be used by default. For example, "ifconfig hme0 metric 1" will to increase the metric of hme0 from its default zero to one.

If an interface is assigned a metric of one when the system is booted, and at a later time another interface is brought up with a metric of zero as in the case of a failover configuration, the private interface must

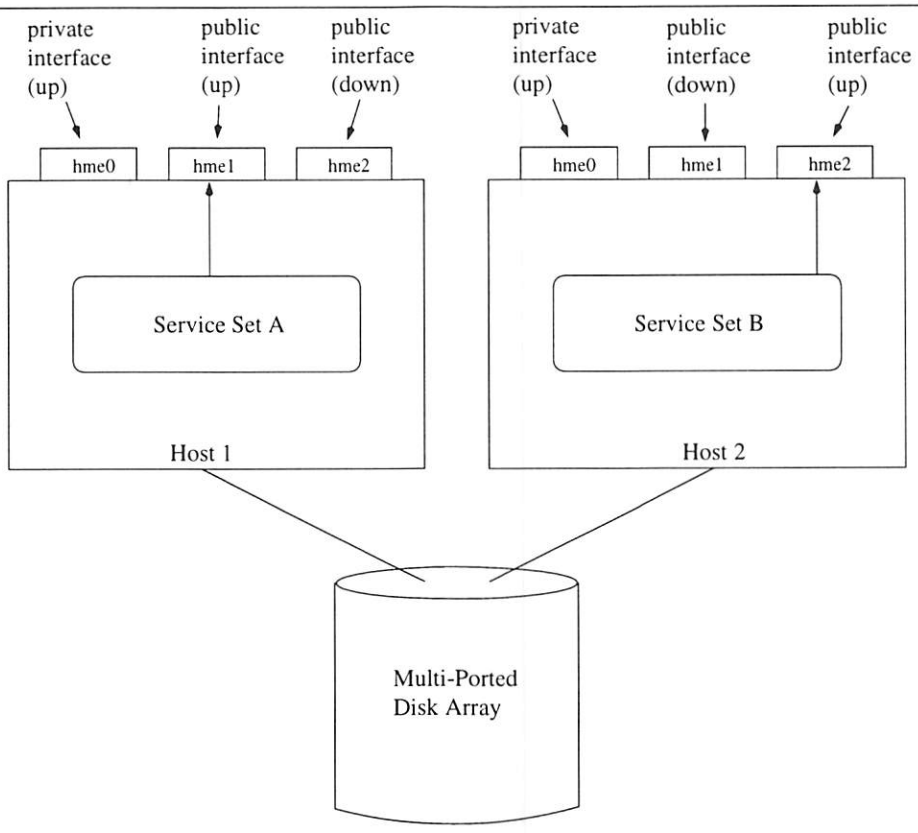

Figure 1: Normal bipolar operating mode.

be taken down and brought back up. Perhaps this will change in a future release of Solaris or as the result of an operating system patch, but Solaris 2.5.1 does not seem to update its routing tables when a second NIC comes up which is connected to the same subnet as the first interface, even if it has a lower metric.

Given a Solaris 2.5.1 or earlier operating environment it is nondeterministic which NIC will be used when multiple interfaces with the same metric are connected to the same subnet. This is generally not a problem when running in bipolar mode since there are only two interfaces up and one was explicitly given preference by setting the metric on the other interface to be higher. However, when operating in combined mode there are two interfaces connected to the same subnet with the same metric. This may cause minor problems for a few services.

Unpredictable choice of interface in this situation should be less of an issue in Solaris 2.6, which provides "interface groups." The Solaris 2.6 kernel maintains a table of IP addresses on which each client has contacted the server. This data is used to determine which IP address to send the reply from [6]. Since clients use the public interfaces to obtain service, the correct interface should be chosen for communication back to the client. The use of interface groups in Solaris 2.6 can be enabled by setting parameter "ip_enable_group_ifs" in the /dev/ip driver to "1" with ndd.

Service Failover

Generic Method

With most services, such as simple web or directory service, doing failover is straightforward. When these services are migrated, it is an easy matter of stopping the service on one host and starting it on the other. If care is taken when performing the migration of services, impact on the clients can be minimized.

The basic algorithm for migrating a service set from host A to B is as follows:

1. Check to see if Host A is still providing the service to to be migrated.
2. If Host A is still providing service set, shell to Host A and:
   a. Stop services which need to be migrated.
   b. Unmount disks which need to be migrated.
   c. Release any locks which may prevent the other host from acquiring the disks to be migrated.
   d. Bring down public NIC associated with service set.
3. Import and mount disks associated with service set being migrated. This should include a check

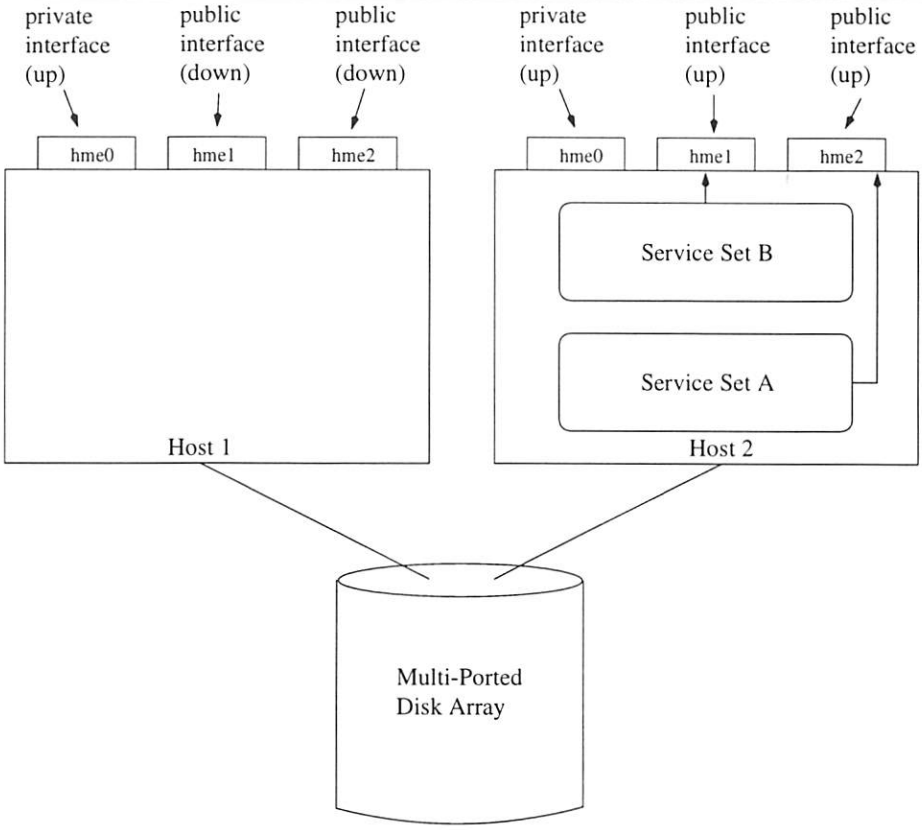

Figure 2: Combined operating mode.

to see if the file system needs a consistency check (i.e., fsck).
4. Bring up public NIC associated with service set.
5. Start services which host A was previously providing.

In our environment, we were able to configure all of our production services in such a way that service migration is possible. However, some services are more difficult to provide within this framework, and others simply cannot be provided in this manner. The following sections describe some of the implementation-specific details for services which are not as straightforward to migrate.

NFS

NFS file handles contain all the information a NFS server needs to distinguish an individual file [7, 8]. In Sun's NFS implementation, this information includes the major and minor device numbers of the file system being nfs exported. Therefore, it is essential the device major and minor numbers of the file system to be migrated be identical on both hosts. If the major/minor device numbers do not match, when NFS service is migrated, the clients will have stale NFS handles.

Either of the two logical volume managers mentioned above will take care of the device names so they are the same on both hosts; however the major device numbers will not necessarily be the same. The relevant entry in /etc/name_to_major for the Veritas Volume Manager 2.3 is "vxio," and for Solstice Disk Suite it is "md." The number associated with this entry should be the same on both hosts.

If the entry for "vxio" or "md" in the /etc/name_to_major file is the same on both hosts, the device files will have the same major and minor device numbers. However, if these numbers are not the same, you will need to edit the /etc/name_to_major file and set one of the numbers listed for "vxio" to be the same as on the other host. Be sure to look through the file to make sure the number you are setting one of the "vxio" entries to is not already in use. After changing this entry in /etc/name_to_major you will need to do a reconfiguration boot. (i.e., boot -r)

Doing NFS service migration with a writable file system is somewhat risky, but in most cases is an acceptable risk given the nature of the situations when failover is needed. When NFS service is migrated from one server to another, any kernel locks (fcntl, flock, etc.) will be lost, while "dot" locks will be preserved. From the perspective of the NFS client, it will look as though the NFS server was rebooted. However, unlike after reboot, the NFS server will not have a list of which hosts which held kernel locks and need to be contacted. It may be possible to preserve kernel locks by migrating the contents of /var/statmon thereby making the NFS server request kernel lock information from the clients as if it had be rebooted.

NFS over UDP handles failover quite well. Assuming there is not a big time gap between when server A stops serving and server B starts serving, the NFS clients do not even notice when service migration has occurred.

NFS over TCP has a few quirks. If NFS services are migrated from server A to server B, the clients "hiccup," complaining "NFS server . . . not responding" and then immediately "NFS server . . . ok." This is caused when NFS clients try to communicate to the failover server which the client believes to have an open TCP stream. The NFS server returns a TCP RST to the NFS client indicating there is no open TCP stream, which causes the NFS client to "hiccup." Upon receipt of the TCP RST the NFS client then sends the appropriate TCP SYN packets to establish a new TCP connection.

A problem occurs with NFS over TCP if you then migrate NFS service from server B back to server A, without either rebooting server A or waiting approximately thirteen minutes between service migration. Even after NFS services are migrated away from a machine and the public NIC is taken down and unplumbed, nfsd maintains an open TCP stream to the nonexistent interface. After thirteen minutes of trying to use the no longer functional interface, nfsd will destroy that TCP stream. If NFS services are migrated back before the nfsd has closed this stream, the NFS clients will hang because of mismatched TCP sequence numbers. This should not be a problem, since in most circumstances where service migration is needed, the duration will be longer than thirteen minutes, or the machine originally hosting the services will be rebooted.

After a migration of NFS services has been performed, one consequence is the data stored in /etc/rmtab is very likely no longer accurate. I have found no negative side effects other than the showmount command giving an incorrect list of hosts.

Another NFS over TCP issue is the case in which your failover-capable NFS server also acts as a NFS client in that it NFS mounts file systems from remote machines. Because the public network interfaces will have a higher preference (a lower metric) than the private interface, the public network interfaces will be used when NFS mounting a remote file system. In the case of a service migration, the public interface being used may be taken down, thereby causing the machine to hang on the apparently downed NFS server. One workaround for this problem might be to temporarily unmount all remote file systems while performing a service migration.

sendmail

Although DNS MX records can be used to provide fallback mail service, this will not provide data availability while the host which spools e-mail is down. Berkeley sendmail 8.8.x can configured to allow for service migration.

In a dual-host configuration it may be desirable to run a sendmail process on each host, but not necessarily configured to perform similar functions – for example, one host may spool mail and the other host may handle off-site delivery. One solution to this problem might be to have three sendmail configuration files: one for host A, one for host B, and another to allow a single sendmail process to fulfill the roles of both machines for operation in a combined post-migration mode.

There is a cleaner solution which allows sendmail to operate on a combined machine using the same configuration files which it uses when running on both hosts. This solution does require a few changes to your sendmail configuration file, a few compile time options, and as of Berkeley sendmail 8.8.7, a small source code modification.

To provide the basis for sendmail service migration, the sendmail binary, configuration file, mqueue, and possibly a mail spool, need to be located on the multi-ported disk array. This may require various compile time options to be set in order to specify an alternate mqueue directory and sendmail configuration file location.

So that sendmail will read its sendmail.cf configuration file and store its sendmail.pid in a nonstandard location, add the flags provided in Listing 1 to "ENVDEF" found in your sendmail makefile. Of course, replace the path names with the ones correct for your system.

Within the sendmail configuration file, you will need to change the default mqueue directory to a path which resides on the multi-ported disk array. To change this, add the following to your m4 configuration file:

```
define('QUEUE_DIR',
  '/mailgate/var/spool/mqueue')
```

For those not using m4 macros, modify the QueueDirectory entry in your sendmail configuration file to reflect the correct path name.

Version 8.7 and beyond of Berkeley sendmail has a configuration option called "DaemonPortOptions." Using this option, you can force sendmail to bind to a specific IP address for daemon mode, which can be used to make sendmail use a specific public NIC associated with the given IP address.

This feature is useful in a dual-host configuration when sendmail services have been combined on one host; specifically, two sendmail processes can be run on one host, both in daemon mode, each binding to a different IP address. This allows sendmail to function the same whether it is one process on each host, or two processes on one host.

To force sendmail to bind to a specific IP address when running in daemon mode, add the following m4 macro configuration file option:

```
define('confDAEMON_OPTIONS',
    'Addr=mailgate')
```

The sendmail configuration file equivalent to this is:

```
O DaemonPortOptions=Addr=mailgate
```

Of course, use the public host name of your mail server instead of "mailgate."

Although sendmail can be easily configured to bind to a specific IP address for accepting incoming SMTP connection in daemon mode, as of Berkeley sendmail 8.8.7 there is no option to force sendmail to bind a specific IP address when establishing remote connections. This has the potential to cause the header "X-Authentication-Warning" to be inserted into e-mail when sendmail connects to a remote host but reports its hostname to be different than the name which maps to the IP address it used for the outgoing connection.

To resolve this outgoing SMTP IP address problem, I found that a one line addition to the sendmail source code can be used to force sendmail to use the same IP address for outgoing connections as listed in the configuration file used for accepting incoming SMTP connections. This sort of functionality is a planned feature for a future release of Berkeley sendmail [9]. This may not be necessary under Solaris 2.6 due to the feature called "interface groups" mentioned above.

In the file daemon.c as obtained from the Berkeley sendmail 8.8.7 source code distribution, find the line:

```
i = connect (s,
    (struct sockaddr *) &addr,
    addrlen);
```

Insert the following right before the connect line you just found:

```
(void) bind(s,
    (struct sockaddr *) &DaemonAddr,
    addrlen);
```

You may want to surround this line by "#ifdef" preprocessor statements for conditional compilation, since this source code modification may not be appropriate for multi-homed hosts on multiple physical networks.

```
ENVDEF=-D_PATH_SENDMAILCF=\"/mailgate/etc/mail/sendmail.cf\" \
       -D_PATH_SENDMAILPID=\"/mailgate/etc/mail/sendmail.pid\"
```

Listing 1: Defining non-standard files.

When running in a combined mode, there is the potential for one sendmail process to try to talk to the other sendmail process. It is desirable for the sendmail processes to communicate as if they were on separate hosts. A modification of the configuration file is necessary so sendmail does not erroneously detect it is talking to itself and disallow it to prevent a mail loop. This can be solved by hard coding macros "$w" and "$j" in the sendmail configuration file to be the unqualified host name and fully qualified hostname (respectively) of the public interface being used. To force correct values into "$w" and "$j", use the following in your m4 configuration file:

```
LOCAL_CONFIG
Dwmailgate
Dj$w.$m
```

If you are not using the m4 macros, just drop the "LOCAL_CONFIG" line and put the "Dw" and "Dj" lines in your sendmail.cf file.

World Wide Web (httpd)

In our environment, doing web service failover was quite simple. Our web site does mostly "vanilla" web file serving with very few cgi scripts.

Most modern web servers have an option which allows you to bind a web server to a specific IP address. For example, the option with the NCSA web server is "BindAddress" found in "httpd.conf." Set this to the IP address or hostname of your public interface through which clients obtain web service.

Because a web server can be bound to a specific interface or IP address, you may run web servers on both hosts in your failover configuration and expect them to operate in a similar manner when running in combined mode.

inetd

Services which are managed by inetd can also be easily migrated from one host to the other. To accomplish this, as many as three separate inetd configuration files may be needed: a generic inetd configuration file listing services provided by both machines, and an individual inetd configuration file for each of the two machines listing only the inetd managed services unique to each machine.

Solaris allows more than one inetd process to run simultaneously, and inetd will accept the name of an alternate inetd.conf file on the command line. Since there is the potential for all three inetd configurations to be running concurrently on one host there may not be any conflicts between the configurations in which more than one service is associated with the same port.

Behind the Scenes

In addition to planning service migration of daemons which provide service to end users, it is important to consider those services which work behind the scenes, such as cron and system backups. The following sections describe how to schedule these events so they happen consistently. If a service migration happens while one of these jobs is running, it may need to be manually restarted after the migration occurs.

cron

It is a relatively simple matter to ensure that cron jobs are executed in the same manner in a combined failover mode as they do when services are spread across both machines. This requires identical crontab entries on both machines where each job is preceded by a test to determine if determine if the given cron job should be run.

For example, if one service set needs a cron job to regenerate a class list daily, a crontab entry might look like:

```
30 0 * * * [ -x
    /opt/class_list/generate ]
    && /opt/class_list/generate
```

The test expression assumes if the program "generate" exists, the service set which requires its use must be running on localhost. The test expression must therefore always test for a file or directory which will only be present when the given service set is running on localhost.

One unfortunate aspect of this is if you have an error in the crontab file, it may escape detection for some time. Errors in the test expression may silently cause a job to not run. If a service set's crontab entry is not correct on both hosts, an error may not cause problems until service migration has occurred.

Backups

Since backups must occur even when services have been migrated, it is important to put thought into the backup software configuration.

In the situation where a remote host performs the backups of both servers, the configuration is simple – just have the software use the public network name of the service set which contains the file systems to be backed up. The backup software should then be able to always find the data to be backed up.

In our environment, both machines have their own local tape drives and run non-commercial backup software. To ensure backups happen consistently, the backup schedules are identical on both machines. This causes the tape indexes to be the same on both tapes, however only a small dump record is written to tape for file systems on the backup schedule which are not present on the host doing the backups.

This backup strategy works well for full level zero dumps, but it may cause problems for incremental dumps. Although both machines will have identical backup schedules, the time stamps listed in /etc/dumpdates for both hosts will be close, but not the same. Therefore, there is the potential to miss data when performing an incremental backup of a file system which

was resident on the other host at the time of the last backup. One possible solution to this would be to write a program to keep /etc/dumpdates files synchronized, and populated with correct time-stamps.

System Reboot

When a host provides a service set, whether it normally resides there or has been migrated from another host, it is important services be resumed if appropriate after a reboot. This issue can be addressed in a boot-up rc script using the following strategy:

For each service set which I may provide,

1. check to see if another host is providing given service set; if so skip item 2 on this list.
2. If the given service set was being provided at the time of system shutdown, start running that service set.

Performing this check on boot-up will ensure that if a machine goes down while hosting a given service set, it will restart the service set after boot-up provided there is no other machine currently offering said service set. If using Veritas Volume Manager, it is simple to determine if a service set was up on a machine by checking for the existence of the directory /dev/vx/dsk/<set_name>.

An Example Implementation

This section contains details about how I chose to implement the failover system in use at Michigan Technological University. It is certainly not a definitive guide on how a failover system must be configured. Use it in the context it is intended – as an example.

The software I wrote for service migration is actually just a small collection of Bourne shell scripts and a short C program. The scripts are written to be simple, effective, and easily configurable.

- acquire – used to migrate given set of services to localhost.
- release – helper script for acquire usually run via rsh instructing the machine providing a service set to stop those services and release the relevant file systems.
- setup_interface – helper script for both acquire and release which is used to configure, bring up, and down the ethernet network interfaces associated with given service set.
- notify_switch – C program required after a service migration if both servers are on separate switched ports on a hub. This simple C program does a broadcast which generates traffic on all its network interfaces and causes the hub to realize the IP/MAC address pair moved to a different port.
- unlockdg – forcibly remove a host lock on a given disk group. This script is useful in the case of an ungraceful service migration where the host which held the lock is down or unable to release the lock.

One way I attempted to keep these scripts simple was to use the name of the public interface to also label the Veritas Volume Manager disk group. Therefore, there is a direct mapping between a set of services, the public host name, and the disk group name. In order to put as little configuration information in the scripts as possible I created a volume called "config" on each disk group which contains configuration information about that particular service set.

The "config" volume contains a small collection of binaries, configuration files, and rc startup and shutdown scripts. This file system gets mounted as /<service_set_name>. The files in "config" are in a similar arrangement to where the normal files are, except with "/" relative to /<set_name>. You might find directories such as /etc, /etc/mail, /etc/dfs, /var/spool/mqueue, /usr/lib, and /sbin containing configuration information essential for the given service set, such as vfstab, dfstab, sendmail.cf, and inetd.conf.

The "acquire" script examines the contents of this configuration file system to determine which file systems to mount, NFS export, which services to start via rc scripts, and if any services require an inetd process to be run for that service set. The release script uses the rc scripts on this file system to stop services which were started via rc scripts.

Since using the rc script to stop services from a given service set may not always kill every process using the file system which needs to be unmounted, the "release" script makes use of the "fuser" command to list processes using a given file system. Processes are first sent a polite SIGTERM, and if still running after one second, they are sent a SIGKILL. If there are any kernel locks on a file system to be unmounted, the lock manager must be stopped before the unmount and started after the unmount.

Increased Availability or High Availability?

While the application of the techniques presented within this paper may give extra flexibility and allow uptime when there would normally be service interruption, this is in no way a HA (High Availability) solution. Operator error, poor fault detection, and failure of the network fabric are just a few items which may cause service interruption. Sites which require a guaranteed percentage of uptime should explore commercial HA solutions which will perform fault detection and automatic failover.

Why stop with manual failover? On the surface it may not seem difficult to write a program that would perform the functions discussed in this paper in an automatic failover capacity. Unfortunately, the problem of doing failover gets considerably more complicated when you intend for the system to run on autopilot and migrate services without any human intervention.

The main problem with automatic failover is one node detecting when the other node is down and

determining if it is a situation in which it should attempt to run the downed node's services. Incorrect detection of failure may lead to inappropriate failovers, which have the potential to bring more instability to your system than homegrown automatic failover can hope to provide.

Availability

Example source code for performing service migration may be obtained from http://www.it.mtu.edu/failover/.

Conclusion

It is possible to build a framework through which service availability can be increased by separating the service from the server. In the event of a server hardware failure or the need for system maintenance, the system administrator can easily migrate the services to a different machine. In addition to increasing service availability, this may also reduce the stress put on a system administrator who, for example, just had a CPU fail in one of the primary production servers.

Of the high availability solutions prevalent today, some provide attractive functionality, but there is still much to be done in this area. As node clustering and single system image become a mature high availability alternative, many of the topics which this paper explores may become non-issues.

Acknowledgments

Thanks to Thomas Dwyer III of Sun Microsystems for his technical guidance with this project and allowing me to use him as a sounding board for my ideas. His insight and suggestions have proven invaluable.

Thanks to my coworkers and my supervisor, Ann West, for supporting this project. Also thanks to the folks who proofread and critiqued my paper – they have certainly helped make this paper more readable.

Author Information

Michael R. Barber is currently a Senior Systems Programmer in Michigan Technological University's Information Technology department, and holds a bachelors degree in Computer Science. You can reach Michael electronically at barber@mtu.edu.

Bibliography

[1] Schemers, Ronald J., "lbnamed: A Load Balancing Name Server in Perl," *Proceedings of the USENIX Systems Administration (LISA IX) Conference*, pp. 1-11, Monterey, CA, September, 1995.

[2] Uniq Software Services, "UPFS – A Highly Available Filesystem," http://www.uniq.com.au/products/upfs/UPFS-WhitePaper/UPFS-WhitePaper-1.html.

[3] "Frequently Asked Questions about Sun NVRAM/hostid," http://www.squirrel.com/squirrel/sun-nvram-hostid.faq.html.

[4] ANSI/IEEE Std 802.3, *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, 1996 Edition, pp. 12-14.

[5] Andrew S. Tanenbaum, *Computer Networks*, Third Edition, 1996, pp 280-281.

[6] USENET post on comp.unix.solaris by Gavin Maltby of Sun Microsystems, 8/22/1997.

[7] RFC 1094: Sun Microsystems, Inc, *NFS: Network File System Protocol specification*, 03/01/1989.

[8] RFC 1813: B. Callaghan, B. Pawlowski, P. Staubach, *NFS Version 3 Protocol Specification*, 06/21/1995.

[9] Personal electronic correspondence with Gregory Neil Shapiro of WPI, 9/2/1997.

The Cyclic News Filesystem: Getting INN To Do More With Less

Scott Lystig Fritchie – Minnesota Regional Network

ABSTRACT

When Usenet News servers were first implemented, the design principle of storing each Usenet article in a separate file appeared to be sound. However, the number of Usenet News articles posted per day has grown phenomenally in the past decade and shows no sign of abating. To stay ahead of the growth curve, Usenet administrators have been forced to buy faster machines, more RAM, and many more disk drives. Many of the performance limitations are caused by interactions with the underlying OS's filesystem, which is usually a Berkeley Fast Filesystem (FFS) derivative.

The Cyclic News Filesystem (CNFS) was designed to avoid most of FFS's major problems when used with INN: synchronous file linking/unlinking and sequential scanning of directory files. Articles are stored within a relative handful of large files, either as regular files on top of a standard filesystem or as block disk devices. Articles are stored sequentially within each file, resuming at the beginning of the file when the end is reached. Disk activity is reduced by an order of magnitude.

Introduction

Though Usenet server software packages have changed greatly over the years, almost all have one implementation detail in common: the method used to store articles on disk. The newsgroup hierarchy maps directly onto a filesystem directory hierarchy, and storing articles in individual files fits the paradigm nicely.

But Usenet article volumes have grown exponentially for at least ten years. Confirmation can be found by reading [Collyer], [Salz], [Swartz93] (whose exponential forecast in 1993 underestimates current 1997 volumes by a factor of four), and by asking any seasoned Usenet system administrator. If that same administrator were asked, "Does your Usenet server have sufficient capacity for current article volume?" the answer would probably be "No" or "Barely." Nothing appears to be on the immediate horizon to slow down article growth rates, which means most Usenet servers will have severe performance problems soon, if not already.

The "one file per article" storage method continues to be used by INN, the most commonly-used Usenet server software in the world today. Most of the operating systems running underneath INN also use a variation of the Berkeley Fast Filesystem (FFS). The FFS, while a significant improvement over its predecessors [McKusick], interacts quite poorly with INN. At today's volumes, an INN server may store thousands to tens of thousands of article files within a single directory. FFS performance degrades quickly when managing large directories, due to sequential directory data scans and synchronous directory updates.

One avenue for addressing this pressing problem is to alter or abandon the current method of storing articles. This paper discusses both. The first stores articles in individual files that are named by an MD5 checksum of their Message-IDs. The other stores thousands of articles within a single file, utilizing the file as a huge cyclic buffer. These buffers can be large files on top of a standard filesystem (even FFS-based) or a block disk device, bypassing standard filesystems entirely. A side benefit of this method is that there is no explicit article expiration process: articles are automatically overwritten as new articles arrive. The resulting performance, as measured by article throughput, increases by a factor of three to four; disk I/O is reduced a factor of 10. There are many nontechnical reasons why the cyclic buffer storage method is preferable to the current method. Most of them are expressed in [StSauver]:

> My throughput is up, my news spool disk I/O bottlenecks have disappeared, I no longer am offline for longer expiration-related periods, I don't need to screw around with software disk striping utilities, and I don't run out of space in /var/spool/news. It's great, and it's free.

The rest of this paper is organized as follows: the next section follows an article through INN's processing and describes how an FFS-based filesystem adds a significant hidden cost to that processing. Then, the INN Improvements section describes previous steps taken to lower that cost. The next two sections describe two of the author's methods to change INN's article storage method to lower or avoid FFS overhead. The Performance section compares the performance of a standard INN server to one utilizing the cyclic storage method. The final sections provide some directions for future INN-related research and some concluding observations.

Life As an Article Processed by INN

INN has separate software components that process an article during its four phases of existence in an INN server. Each phase has problematic interactions between INN and FFS. They are as follows:

1. Receive the article and store it in a file in the spool directory.
2. Read the article file from the spool directory to forward it to peer servers.
3. Read the article file from the spool directory to forward it to NNTP reader clients.
4. Remove the article file from the spool directory.

An Article Is Received

INND is the central daemon process that accepts or refuses articles offered via the NNTP protocol [Kantor], stores the article, keeps a database of articles it has already processed (in `history` and related DBZ index files `history.pag` and `history.dir`), and decides which peer(s) the article should be forwarded to.

See Figure 1 for an example article that INND has just accepted. INND consults the `active` file to assign this article the next available number for "misc.jobs.offered", which could be 9124816, and stores the article in a file with the path `misc/jobs/offered/9124816` (relative to the SPOOLDIR directory, e.g., `/var/spool/news`). When storing the sample article, the open() system call will trigger a behind-the-scenes scan of several directories first. (See Figure 2.)

The directory name cache may eliminate the need to fully scan directories 1-6, and the file buffer cache may avoid the need for some or all of the disk I/O. Articles tend to arrive with a fairly random newsgroup distribution, which makes the OS caches less effective without large amounts of RAM to assist them. In the worst case, directories 1-6 must be scanned sequentially to find the inode for directory 7, reading 194KB of directory data; each scan would trigger at least one disk I/O.

Once the inode for directory 7 is found, it must be read sequentially to look for an unused space to insert file `9124816`'s directory entry. If there is no unused space, then the entry is appended to the end of the directory file. A quick analysis by the author suggests that an average of 56% of the final directory must be scanned before insertion is possible.[1] On average, 357KB of directory data is read before finding a suitable insertion point. (Both [Sweeney] and [Rakitzis] illustrate the wastefulness of linear scanning of large directories.) Furthermore, the insertion operation is synchronous: the change must be committed to disk before the open system call can return.

As a result of sequential directory scanning and synchronous file linking, the amount of time necessary for INN to write a single article file, regardless of the its size, can vary tremendously. Almost every piece of

[1] An unattributable Net rumor estimated 70%. The author's test sampled the `misc/jobs/offered` subdirectory every five minutes for a 24 hour period.

```
Path: news.mr.net!mr.net!visi.com!news-out.visi.com!ix.netcom.com!news
From: John Doe <jdoe@no.where>
Date: 2 Sep 1997 18:16:52 GMT
Newsgroups: misc.jobs.offered,mn.jobs
Message-ID: <foo87692.bar85-97@no.where>
Subject: French Fry Slicer (Level II) wanted for part-time work
Lines: 22
```

Figure 1: Sample headers from a sample Usenet article.

Level	Directory Name	Size	Number of Entries
1	/	512B	31
2	/var	512B	20
3	/var/spool	512B	12
4	/var/spool/news	15KB	994
5	/var/spool/news/misc	3KB	127
6	/var/spool/news/misc/jobs	174KB	10,836
7	/var/spool/news/misc/jobs/offered	637KB	32,499
5	/var/spool/news/alt	25KB	1367
6	/var/spool/news/alt/atari	512B	1
6	/var/spool/news/alt/fan	15KB	771
10	/var/spool/news/alt/swedish/chef/bork/bork/bork	512B	3

Figure 2: Sample directory sizes for an INN server spool filesystem

hardware in the server affects this measurement, though the number of disk drives spanned by the filesystem, the spanning type (e.g., none, RAID 0), the filesystem type, and the overall disk I/O workload of the drives are the largest factors influencing this measurement. It is not unusual to witness write times below 10 ms to over 200ms. INND, which is a single-threaded process, is blocked until open() is finished.

It should be noted that the sample article was cross-posted to another newsgroup, mn.jobs. INN will use a hard or symbolic link for `mn/jobs/8149` to link to the original article file, `misc/jobs/offered/9124816`. The linking process is identical to the file creation process with respect to directory file modification. Again, INND blocks during the link system call.

Another task INND performs is the processing of control messages. Article cancellation messages are by far the most frequently encountered control messages. To process a cancel message, INND performs a `history` database lookup to check if the message to be cancelled has arrived yet. If it has, the article is removed by INND. The amount of time this operation takes can vary from 8 ms to over 250 ms. Unlink() is also performed synchronously under FFS, which yet again halts INND's activity until the directory data is committed to disk.

Some Usenet servers are already seeing article volumes at or above 750K articles per day. A server must accept, on average, 8.68 articles/second, or 115 milliseconds/article, just to stay even with traffic flow. The 115 milliseconds includes network latency, server overhead (`history` database queries are the biggest INND factor here), and storage subsystem overhead. The measurements in Figure 8 and Figure 9 suggest that filesystem overhead and disk latency account for, at a bare minimum, an average of 50ms of overhead per article. The true value is probably closer to 75ms, and that's on a well-configured and otherwise idle server! Continued exponential article growth rates will only make this problem worse, particularly with a single-threaded server application.

An Article Is Forwarded

INND's `newsfeeds` file contains the rules for determining whether an article should be forwarded to one or more peer servers. Programs such as INNXMIT, INNFEED, and NNTPLINK are used to perform the actual article transmission.

INNXMIT reads a batch file containing the Message-IDs and filenames of articles to be forwarded to a particular Usenet peer server; there is at least one INNXMIT process per feed.[2] INNXMIT processes are run periodically by a shell script called "nntpsend", which is usually run via "cron".

Once INNXMIT has made an offer to send an article to a peer and the peer has accepted, INNXMIT's open() of the article triggers another flurry of OS activity. If lucky, the OS need not trigger any disk I/O because all of the required data is stored in-memory in the directory name and filesystem buffer caches, though the kernel CPU time spent searching the latter may be non-trivial. In the worst case, the example will require a linear scan of 830KB of directory data, triggering several disk I/O's to different filesystems. INNXMIT's batch file is written in the same order in which INND received the articles, adding a significant amount of randomness that renders the OS caches less effective.

The roles of NNTPLINK and INNFEED will be addressed in a later section.

An Article Is Read By Reader Clients

When INN was originally written, as with C News and its predecessors, Usenet client software accessed articles by reading the article files directly out of the spool directory. For a client such as "readnews" or "rn" to see what articles were available in "misc.jobs.offered", it simply had to generate a list of files in the `misc/jobs/offered` subdirectory. Usenet users had accounts on the Usenet server; client software ran on the server.

Today most Usenet reader clients use the NNTP protocol to access Usenet articles. INN includes a separate program, NNRPD, to handle communication with NNTP reader clients. Like many other UNIX-based servers, INND will fork() and exec() a child NNRPD process for each simultaneous reader session.

NNRPD has the same filesystem-related problems that INNXMIT does, though to a lesser degree. Filesystem buffer and directory name cache hit rates are better because NNRPD is much more likely to access a large number of article files in the same directory, avoiding cache churning. Also, FFS's tendency to group files in the same cylinder group also means that disk heads do not usually travel far to read files in the same subdirectory. Unfortunately, the heads are quickly relocated by other server activity, since most NNTP reader clients are run by humans who ordinarily take more than a few milliseconds to read a typical Usenet article. "Sucking" clients and automatic binary decoding utilities are growing in popularity, generating larger bursts of intensive disk operations than purely human-controlled software.

Another significant problem is that each article file access triggers an update of that inode's "last access time" value, which requires a disk write operation after each file is read. (The update also occurs when INNXMIT, NNTPLINK, or INNFEED reads a file to be forwarded to a peer.)

[2] An INN administrator may configure multiple feeds to a single peer, e.g., one feed of articles under 100KB in size and one for articles over 100KB.

An Article Is Removed

An article can be removed in several ways:

1. removed by a cancel control message.
2. removed by INN's "expire" process, usually via the "news.daily" script.
3. removed by a third party, e.g., an administrator using "rm -rf /var/spool/news/alt/binaries".

"Expire"'s list of files to remove is usually sorted, so close locality of reference helps achieve good cache hit rates. However, the final directory search is a sequential one (barring assistance from the directory name cache), requiring significant kernel CPU time to process. Each unlink() operation is synchronous, slowing the operation down much further: unlink() can take anywhere from 8ms to over 300ms to complete. If the article is cross-posted, all corresponding hard or symbolic links must also be removed (each at identical cost).

"Expire" is usually run early in the morning, when CPU utilization is lower, fewer articles are being transferred in/out (feeds across timezones can reduce this advantage), and fewer NNRPD processes upset the spool filesystem with additional disk activity. While "expire" is running, however, the disks(s) in the spool filesystem are effectively saturated with I/O requests: additional operation, such as sending/receiving articles, only slows down both the article expiration and the sending/receiving processes even further.

Improvements in INN Performance, a.k.a. Previous Work

A number of changes have been made to INN over the years to improve its performance. The following is a partial list of the more important ones:

1. Sort the list of article files to be removed and use "fastrm" to remove them.

   Sorting tremendously improves locality of reference in OS caches by removing all designated files in, for example, `misc/jobs/offered` at once rather than removing them in the order in which they arrived. Arrival order is typically quite random, which causes poor file buffer and directory name cache hit ratios when removing article files in arrival order. "Fastrm" first changes the working directory to the one containing the specified files; it calls chdir() using a relative path if shorter than a full path, and it gives simple filenames only to unlink(), which lowers namei() overhead.

2. Use the "-L" INND flag and "crosspost".

   The program "crosspost" performs the linking operations for cross-posted articles that INND would otherwise do. It doesn't accelerate the underlying filesystem operations, but it keeps INND from excessive blocking while waiting for the links to be made.

3. Mount multiple filesystems under spool directory.

   Originally done to provide the spool directory with more storage capacity than any single disk drive, it also provides a method of distributing disk I/O over multiple disk drives and disk controllers.

4. Make the spool directory a single virtual filesystem spanning multiple disk drives.

   Anyone who has tried to run an INN server receiving a full feed onto a single spool disk drive will discover that there aren't enough hours in a day to perform all the disk activity an FFS filesystem requires. Virtual filesystems can be implemented in software, e.g., Sun's Solstice DiskSuite or *BSD's and Linux's concatenated disk drivers (ccd and md, respectively), or in hardware, e.g., hardware RAID controller. This method has an additional benefit of creating a single pool of disk space, making it less likely that a burst of articles will fill the filesystem than improvement #3, as well as eliminating the need to use symbolic links across filesystems.

5. Sort INNXMIT batch files by filename instead of arrival order.

   Sorting INNXMIT batch files (in the "nntpsend" script) allows OS caches to be more effective for the same reasons sorting "expire"'s batches improves file removal rates. The author has experienced up to three-fold improvement in small article transfer rates by INNXMIT when using sorted batches, particularly if the batch is large and contains relatively few unique newsgroups. Experimental data can be found in [Delany].

6. Use NNTPLINK instead of INNXMIT.

   NNTPLINK runs concurrently with INND, one NNTPLINK process per peer feed. If the peer can accept articles quickly enough, NNTPLINK will read an article file within a second of being written. In this case, the odds are quite high that the OS still has all necessary information in its caches, avoiding the need for disk I/O. It also improves directory name cache performance, which in turn lowers directory scanning requirements.

7. Use INNFEED instead of INNXMIT.

   INNFEED operates similarly to NNTPLINK, but a single INNFEED process can feed multiple peers. Articles are read in their entirety into memory, then reference counts are used to avoid re-reading articles to be sent to multiple peers. A single process also helps reduce VM and context-switching workloads.

8. Avoid inode last access time updates.

   Linux users can use the "no_atime" mount option, and Network Appliance users can use

"option no_atime_update" [Swartz96] to avoid updating an inode's last access time attribute each time the file is read. Does a Usenet administrator really care about the last time an article file was read?

9. Use the "async" mount option.

   BSD users can mount their spool directory with the "async" option, which causes synchronous file operations to be performed asynchronously. Performance can improve dramatically but the risk of filesystem corruption in the event of a system crash is orders-of-magnitude higher. Linux's ext2fs uses asynchronous metadata updates by default; its careful ordering of disk writes greatly reduces the risk of filesystem damage. Legato's Prestoserve card negates this risk by making metadata updates into NVRAM, giving asynchronous-like behavior, then committing the changes to disk at a later time.

10. Use a dedicated, high-performance fileserver for spool storage.

    Use of high-performance fileservers such as Network Appliance can greatly reduce I/O delays [Swartz96] as well as serve the spool filesystem to multiple machines [Christenson]. Negative aspects include NFS and TCP/IP protocol overhead as well as potential network congestion and latency.

11. Use non-FFS-based filesystems.

    The XFS [Sweeney] and WAFL [Hitz] filesystems offer a number of enhancements that help minimize directory scanning requirements. Both, if the entire Network Appliance toaster is also considered, also minimize metadata update latency via write-ahead logging to disk or to NVRAM, respectively.

12. Multi-threaded NNRPD.

    Like INNFEED, a multi-threaded NNRPD reduces VM and context-switching requirements. An implementation in Java is already available [Poskanzer].

13. Avoiding use of INN altogether.

    Several Usenet software packages boast large performance increases compared to INN running on identical hardware. These include Diablo [Dillon], NNTPRelay [Sedore], KNews [Krten], and Cyclone [Highwind]. All of these packages avoid the standard "one article per file" storage method.

One innovation, the "streaming" extensions to the NNTP protocol, has not been formally defined, though it is widely used. It's primary benefit is reducing network-related latency in the non-streaming "IHAVE" lock-step dialog. If a server's disk subsystem is already at or near saturation, reducing network latency will only push drive utilization even higher.

INN Solution #1: File Naming by MD5 Hash

Perhaps first proposed in [Aguirre], this method was suggested to reduce file creation, access, and removal delays by making the INN spool directory "bushier," i.e., putting a limit on subdirectory depth and limiting the number of files in any particular directory. (With a standard INN server, it's common to run into directory depth extremes like those in Figure 2.) Instead of storing an article in a path based on a newsgroup + article number tuple, the Message-ID is hashed using the MD5 cryptographic hash algorithm [Rivest]. The result is converted to an ASCII hexadecimal representation and then modified using this algorithm. (See also Figure 3.)

1. Use 6 bits in the first byte as the first subdirectory name.
2. Use 6 bits in the second byte as the second subdirectory name.
3. Use the next 4 bytes for the filename.
4. In the event of a collision, append a "+" to the filename and test again for collision.

This method guarantees that any file's path would contain exactly two subdirectories and that each directory has exactly 64 subdirectories.

In April 1996, when this method was implemented, Usenet article volume was typically 180,000 articles/day. Subdirectories on a feeder machine storing articles for three days would contain approximately 125-135 files. Each filename was 8 bytes long, making each subdirectory consume about 2KB.

The MD5 hashing experiment taught the author a number of valuable lessons:

1. Article numbers are not required for feeder-only INN servers.
2. The standard `news/group/854` pathnames can be transformed into opaque "where is the article stored?" strings without affecting too many of INN's components.
3. Due to existing "history" and "expire" implementations and a desire not to meddle with those implementations, only a single expiration policy is feasible, instead of multiple per-hierarchy and per-group policies.

Message-ID	<foo87692.bar85-97@no.where>
MD5(Message-ID)	a0c277ebdad179a67c0203de8295269b
File path	20/02/77ebdad1

Figure 3: MD5 hash storage example

4. Disk I/O utilization, compared to a standard INN server on the same hardware, is cut roughly in half.

The MD5 hash version of INN was used in production on the Minnesota Regional Network's (MRNet's) three feeder-only machines for a period of about two months in mid-1996. Though the source code wasn't widely-distributed, it was used by at least one server in the Usenet Top 10 [Top1000] until approximately May of 1997.

INN solution #2: Article Storage in Cyclic Buffers

Filesystem research is an extremely active field of study. All UNIX filesystem improvements, from small changes in existing implementations to entirely new ones, must include functional characteristics such as the ability to create, modify, and remove files and their associated metadata, manage directory hierarchies, access control, concurrency restrictions, access auditing, and so on.

As pointed out in [Templeton96b], most of those characteristics do not apply to Usenet articles. Specifically, articles sizes are known in advance and do not grow during creation. Article text is never modified once committed to disk, simplifying space management policies. In addition, Usenet articles are stored for relatively brief periods of time, usually under two weeks, and large batches of articles, which arrived at similar times, are removed at the same time. If a Usenet administrator must store articles for longer, the number of articles is low enough that the large directories aren't a large problem or the articles are stored using an entirely different method, e.g., [DejaNews].

Cyclic Article Storage Method

The cyclic buffer article storage method was proposed in [Templeton96a]. When an article is accepted, the server decides which cyclic buffer will store the article. The server may choose between one or more buffers. The decision algorithm might consider the article's size, posted newsgroup(s), the "From" address, and/or other criteria. The article is written to the cyclic buffer at the location indicated by the "free" pointer. If the article doesn't fit within space pointed to by "free," the "free" pointer is reset to the beginning of the buffer before writing the article. After the article was written, the "free" pointer advances to a position beyond the end of the newly-written article.

In Figure 4, articles A-L were written inside the cyclic buffer. When the server decided to store article M within the buffer, it moved the "free" pointer back to the buffer's beginning before continuing. The "free" pointer points to the end of article P, the last written into the buffer. Article I is the oldest article still available in the buffer: articles M-O have overwritten A-G, and article P has partially overwritten article H.

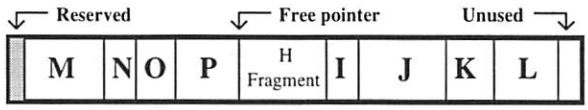

Figure 4: An example cyclic buffer

An INN Cyclic Buffer Implementation

The implementation described below is still considered experimental, though it has been in use at MRNet for over a year as well as on a couple dozen servers world-wide. The main design principles were simplicity and minimization of coding time. INN was chosen as a base to work upon, and few additional bells and whistles have been added. Support for NNTP reader clients was originally omitted from the design, though it has since been added.

The current source code is full of references to "raw disk partitions." When the author first began development, he naively assumed that "raw"/character disk devices would be the ideal storage medium, but their synchronous write semantics result in terrible performance. A "cooked"/block disk device (if the OS supports mmap()'ing block devices) or a large file, e.g., 2GB on top of a standard filesystem, works much better by allowing the OS to delay and consolidate writes. The word "raw," however, will remain until the code is rewritten.

Cyclic INN Article Processing

Once INND has accepted an article, it consults the article storage rules table. Its external, human-readable version is found in a new configuration file called "config.rawpart" (parsed at startup time); see Figure 5, section 3. The first rule matched is used. For the example article in Figure 1, the article will be stored in a "metarawpart" called SMALLARTS.

A "metarawpart" is a collection of one or more "rawpart," or cyclic buffer, components. If a metarawpart has more than one component, the server will choose one of them in round-robin fashion. See Figure 5, section 2 for configuration syntax.

Figure 5, section 1 lists three "rawpart" buffers. In this example, SD0 resides on an old Seagate Hawk 2GB drive. SD1 and SD2 reside on newer Seagate Barracuda 2GB drives. All three are partitioned identically, with one 2GB partition on slice zero. The assignment of large and heavily-cross-posted articles to a 2GB space and smaller articles into a 4GB space reflects the administrator's desired article retention policies. Articles written inside BIGARTS will recycle a couple of times per day, whereas articles written inside SMALLARTS will not be overwritten for many days.

Cyclic Buffer State Information

INND maintains an array of RAWPART structures that stores the state of each cyclic buffer (see Figure 6). While running, INND periodically converts the rawpart state data to ASCII and writes a RAWPARTEXTERN structure to the beginning of each buffer file to preserve buffer state information in the event of a process or system crash. No special efforts are made to synchronize the in-memory state data with the data on disk. It is therefore possible to lose a small number of articles in a crash: some may be overwritten when INND restarts. Similarly, only one copy of the data is written; there is nothing analogous to a "backup superblock." This lack of redundancy has yet to cause a loss of state data in over a year of use.

The state data is written to disk in ASCII in a gesture toward platform-independence. This facilitates serving cyclic buffers via NFS to heterogeneous servers and copying buffers between servers.

Near the beginning of the buffer, between the RAWPARTEXTERN structure and the start of the actual article storage area, is a bitmap that records which 512-byte blocks within the buffer contain valid RAWARTHEADER structures. The bitmap is mmap()'ed to provide easy pointer access to its contents. The amount of space the bitmap uses, e.g., 512KB for a 2GB file, is small. In return, the bitmap makes it possible to determine if an article has been cancelled without attempting to read and validate its RAWARTHEADER structure.

The absence of the bitmap does not adversely affect a feeder-only INN server. Earlier implementations demonstrated that the additional I/O used to attempt to read cancelled article files is not significant. The article's RAWARTHEADER structure is overwritten with the string "CANCELLED," which foils further attempts to read the article. The bitmap's role in the NNTP reader client support, as discussed later, solves an important performance problem.

INND gives an article, once it is accepted, a filename for use within other parts of INN. Unlike a standard filename such as `misc/jobs/offered/9124816`, the cyclic name looks like "SD1:24da800:1e". Colons separate the "rawpart" buffer name, offset within the buffer, and buffer's current cycle number. Together with the information in `rawpart.config`, this name contains all of the information required to locate the article's text. If the rawpart's current "free" offset and cycle number are known, it is trivial to determine if the article has been overwritten by a later article. In this example, the RAWARTHEADER structure would be written at offset 0x24da800, followed immediately by the full text of the article.

A compile-time option determines if the article text is stored with the UNIX-style newline convention or stored in "wire format," with CR-LF newlines and periods at the beginning of a line escaped with an additional "." [Kantor]. Storage in wire format eliminates the need to convert newline styles each time the article is transmitted via NNTP, reducing user CPU time utilization in busy servers.

```
# Section 1
# Format: "metarawpart" (literally) : name : "I" (literally) :
#             comma-separated list of "rawpart" buffers
metarawpart:BIGARTS:I:SD0
metarawpart:SMALLARTS:I:SD1,SD2

# Section 2
# Format: "rawpart" (literally) : name : path : size (in hex!)
rawpart:SD0:/var/spool/dsks/c0t0d0s0:7A000000
rawpart:SD1:/var/spool/dsks/c0t0d0s1:7A000000
rawpart:SD2:/var/spool/dsks/c0t0d0s2:7A000000

# Section 3
# Format: "groups" (literally): rule or wildcard pattern: metarawpart name
# Store in BIGARTS if 1st group matches *warez* or *binaries*
groups:*warez*:BIGARTS
groups:*binaries*:BIGARTS
# Store in BIGARTS if article size is greater than 100KB
groups:~>100000:BIGARTS
# Store in BIGARTS if article is cross-posted to 10 or more groups
groups:~G10:BIGARTS
# Store all others in SMALLARTS
groups:*:SMALLARTS
```

Figure 5: The storage rules section of "config.rawpart"

NNTP Reader Support

The original cyclic buffer INN implementation did not support NNTP reader clients: the reader client reliance on article numbers greatly complicates an otherwise straightforward design. However, MRNet's main NNTP reader server, news.mr.net, started to show its article handling capacity limits in early 1997 – and the NNTP reader support was born.

INND is a busy process, so putting article number support back into INND was the last thing the author wished to do. Unfortunately, INND maintains low- and high-article values for each group inside the `active` file. Many NNTP reader clients rely on those values to be correct; incorrect values mislead many clients into believing that new articles have not arrived in a newsgroup when, in fact, new ones may be available.

A compromise was finally implemented. Most NNTP reader server administrators use "overchan", a program which creates Overview article summaries to eliminate the need for NNTP readers to retrieve the headers of all articles within a group (e.g., for discussion threading). "Overchan" was given the task of assigning article numbers. It also updates the mechanism that maps newsgroup + article number tuples => article storage locations and informs INND of newsgroup high-article value changes so INND can update the `active` file accordingly.

Several databases for the newsgroup + article number tuple => article storage location mechanism, such as GDBM and Berkeley DB, were considered. All were rejected in favor of a simpler mechanism: flat ASCII files with fixed-length lines.

Each newsgroup has an Overview file with a name like `misc/jobs/offered/.overview` (relative to the OVERVIEWDIR directory). A

```
/* Main internal data structure */
typedef struct {
  char          magic[RAWMASIZ];/* Magic string RAW_MAGIC */
  char          name[RAWNASIZ]; /* Symbolic name: 15 bytes */
  char          path[RAWPASIZ]; /* Path to file: 63 bytes */
  RAWPART_OFF_T len;            /* Length of writable area, in bytes */
  RAWPART_OFF_T free;           /* Free offset (relative to byte 0!) */
  struct metarawpart *mymeta;   /* Pointer to my "parent" metarawpart */
  time_t        updated;        /* Time of last update to header */
  int           fdrdwr;         /* O_RDWR file descriptor, "innd" use only! */
  int           fdrd;           /* O_RDONLY file descriptor for this rawpart */
  CYCLENUM_T    cyclenum;       /* Number of current cycle, 0 = invalid */
  int           magicver;       /* Magic version number */
  int           articlepending; /* Debug flag: article is pending for write */
  caddr_t       bitmap;         /* Bitmap for article in use */
  RAWPART_OFF_T minartoffset;   /* The minimum offset for article storage */
} RAWPART;
extern RAWPART rawparttab[MAX_RAWPARTS];

/* Main data structures written to disk */
typedef struct {
    char        magic[RAWMASIZ];
    char        name[RAWNASIZ];
    char        path[RAWPASIZ];
    char        lena[RAWLASIZ];         /* ASCII version of len */
    char        freea[RAWLASIZ];        /* ASCII version of free */
    char        updateda[RAWLASIZ];     /* ASCII version of updated */
    char        cyclenuma[RAWLASIZ];    /* ASCII version of cyclenum */
} RAWPARTEXTERN;

typedef struct {
  long          zottf;          /* This should always be 0x01234 */
  long          size;           /* Article size, converted by htonl() */
  char          m_id[64];       /* 63 bytes of Message-ID should be enough */
} RAWARTHEADER;
```

Figure 6: Important on-disk and in-memory data structures

newsgroup + article number => storage location mapping file called `name.map` is created in that same directory. The example in Figure 7 shows that the first map entry begins at byte 102, the low article number is 817053, the file was last updated on 2 September 1997, and that the group has at most three articles (which may have been cancelled or overwritten).

```
BodyStart    102
LowArt       817053
LastRewrite 873194497
SD1:10a2a000:2a
SD2:1b9b2800:18
SD1:11527400:2a
```

Figure 7: A sample "name.map" file for `misc.jobs.offered`

The `name.map` file format makes additions or changes easy. Each line in the `name.map` file is exactly 34 bytes long (including newline), making offset calculations for an individual map entry simple. NNRPD, when it receives a "GROUP" command, mmap()'s the newsgroup's `name.map` file for pointer-based access to the file. The file is then read, and the appropriate bitmaps are checked to determine which articles still exist; this replaces the readdir() loop a standard NNRPD process uses to create the "ARTnumbers" array of existing articles.

Ordinarily it is sufficient to know only the low- and high-article values for a newsgroup. However, it is necessary in one situation to know exactly which articles have been cancelled: when sending Overview information to the client. If all Overview data between article numbers "low" and "high" are sent to the client, the client believes that all articles in between are accessible. The user is shown on-screen that a cancelled article is accessible, but an attempt to retrieve the article text will fail. MRNet's users considered this to be a bug, not a feature.

Early implementations required reading each RAWARTHEADER structure to determine if an article had been cancelled, which generated an enormous disk I/O burden and caused huge delays in NNRPD's Overview responses. The bitmap was added to each "rawpart" buffer to solve this problem.

Like the Overview's `.overview` files, the `name.map` files require periodic pruning to remove references to articles that have been expired (standard INN), cancelled, or overwritten (cyclic INN). A Perl script called "expire.mapprune" performs this task; like "expireover", it is typically run once per day. To update the file without rewriting it, the "BodyStart" line is simply overwritten with a larger value.

Performance Observations and Evaluation

Until now, most of the evidence of a cyclic INN server's superior performance has been anecdotal. Examples include:

- Use of "sar" and "iostat" to watch disk activity. MRNet's central feeder machine, when using a standard INN server, was engaged in about a dozen bi-directional feeds that kept its disk drives anywhere from 20-50% busy on a sustained basis; during article expiration periods, drive activity was sustained at 70-90% busy. A cyclic INN server and 70 one-way feeds can rarely sustain a 25% busy workload.
- A Pentium 120MHz PC with 128MB RAM, one SCSI controller, three 5400 RPM disks (two for article storage), and a 100Mb/s Ethernet interface is configured with 45 one-way feeds to MRNet members. Several times a member's Usenet server has been down for several hours, and when the server is back online, MRNet receives telephone calls from the members trying to trace a denial-of-service attack. The FreeBSD PC, resuming the Usenet feed, has been the source of the "attack."
- A Sun SPARCstation 5 with an 85MHz CPU, 192MB RAM, one SCSI controller, three 5400 RPM disks (two for article storage), and a standard 10Mb/s Ethernet interface was able to climb as high as 20th place in the Usenet Top 1000 rankings [Top1000].
- The performance improvement in cyclic INN has allowed MRNet to avoid buying faster equipment for Usenet servers for over a year while providing superior Usenet feeds to its members and customers ... with the periodic exceptions of using code that wasn't quite as stable as it should have been.

Most of the observations and measurements in this section are limited to INND's operation because its function is the easiest to measure and to control tightly. The measurements aren't intended to be a rigorous investigation into a cyclic INN's performance; rather, they are a first attempt to quantify how much this implementation lowers disk I/O activity.

Experimental Platforms and Methodology

In an attempt to systematically measure system performance while INND accepted articles, four separate batches of articles were sent to two INN servers. The servers were idle, with the exception of INND's activity: all INNXMIT, INNFEED, NNRPD, and other non-INN-related processes were killed. The `history` file was truncated to zero bytes, followed by a "makehistory -i -r -s 500000"; a large `history` DBZ index file can introduce a significant amount of latency which would skew the test results. The reference INND process was run with the "-L" flag, and "crosspost" was not used to perform the additional linking work. Each server was monitored for CPU utilization, file buffer and inode cache hit rates, and disk utilization using a custom script based on an example from the SE Performance Toolkit [Cockcroft]. The results of all four batches were averaged and shown in Figure 8 and Figure 9.

The reference INN server is a Sun SPARCstation 10 with two 100MHz hyperSPARC CPUs, Solaris 2.5.1, 272MB RAM, a 100Mb/s Ethernet interface, and eight 7200 RPM disks spread across three SCSI controllers. The five disk drives storing the spool filesystem are 4GB Seagate 15150W ("Barracuda 4" Fast & Wide SCSI) all connected to the third SCSI controller, a Sun Fast/Wide SCSI SBus controller. The cyclic INN server is a Sun SPARCstation 5 clone with a single 85MHz microSPARC II CPU, Solaris 2.5, 128MB RAM, built-in 10Mb/s Ethernet interface, and two 5400 RPM disks on the built-in SCSI controller. A single 2GB Seagate ST32430N ("Hawk 2LP" Narrow SCSI) stored the server's single cyclic buffer, which is accessed via a block disk device (rather than a file on top of a UFS filesystem).

The server sending the articles is an Intel PC with a single 200MHz Pentium Pro CPU, FreeBSD 2.1.7.1-RELEASE, 256MB RAM, a 100Mb/s Ethernet interface, and three 4GB 7200 RPM disks on two SCSI controllers. Two of the drives, Seagate 15150Ns, store three cyclic buffers consisting of 2GB files stored on top of two FFS filesystems. The batches of articles it sent were from 5,300 to 5,800 articles each, posted to approximately 3,000 different newsgroups. Each article was less than 32KB in size in order to emphasize article throughput; the average article size was about 2.6KB. The sending machine was under a relatively heavy load: INN was accepting articles from 95 to 100 NNTP sessions, and three INNFEED processes were feeding articles back to 38 peer servers. INNXMIT was used to transmit each batch in a single streaming NNTP session.

The article throughput results in Figure 8 are dramatic. The cyclic INN server accepted articles three to four times faster than a standard INN server, despite the fact that the standard INN server had more CPU, RAM, disk, and network resources available.

The data in Figure 9 are even more dramatic. Spool disk activity across the board is reduced by approximately an order of magnitude. When the facts that there is no explicit time- and resource-consuming article expiration process and that disk I/O for article

Batch number	1	2	3	4	Total
Articles in batch	5,301	5,318	5,826	5,407	21,852
Unique newsgroups in batch	2,961	2,962	2,895	2,616	-
Size of batch (KBytes)	13,537	12,193	15,485	15,571	56,786
Elapsed time by standard INN (seconds)	480	486	551	530	2,047
Articles/second accepted by standard INN	10.1	10.1	9.9	9.4	-
Elapsed time by cyclic INN (seconds)	153	168	135	141	597
Articles/second accepted by standard INN	31.7	29.3	40.3	35.2	-
Article throughput increase factor	3.13	2.89	4.08	3.76	-

Figure 8: Batch article transmission results.

Statistic	Standard INN	Cyclic INN	Δ Relative to Standard
Average CPU idle time	26.41	38.43	46 %
Average CPU user time	13.39	40.09	199 %
Average CPU system time	12.63	12.71	1 %
Average CPU wait time	47.57	8.77	-82 %
Read() + readv() calls	192,280	28,622	-85 %
Write() + writev() calls	367,806	136,368	-63 %
Pathname lookups	35,325	3,606	-90 %
Ufs_iget() calls	26,592	65	-99 %
Spool disk data read (KB)	153,575	50,116	-67 %
Spool disk data written (KB)	519,802	75,688	-85 %
Spool disk reads issued	25,258	1,245	-95 %
Spool disk writes issued	88,600	8,059	-91 %
Spool disk average service time (ms)	48	13	-73 %
Buffer cache hit rate	92.63	98.37	6 %
DNLC hit rate	65.74	93.01	41 %
Inode cache hit rate	11.77	16.20	38 %

Figure 9: Kernel statistics taken during article transmission.

cross-post links was not measured are taken into account, the decrease is more than a factor of 10.

The file buffer and directory name cache hit rates for the standard INN server are quite good, which is consistent with it having a large amount of RAM available and with the system performing no other substantive tasks while the measurements were taken. The amount of data the cyclic INN server read from the spool disk is conspicuously high; though article cancellations and reads of the mmap()'ed buffer bitmap play a role, further study is required to find a satisfactory explanation.

In a less-tightly controlled test, a transfer of a batch of 20,000 articles, all 4KB in size or less, between two Sun SPARCstation 5s which both use cyclic INN, exhibits sustained transfers of 100-150 articles/second (when the `history` file is first truncated) and occasionally peaks to 200 articles/second. Though this test was not performed on the reference INN server, the author would not anticipate that the server would exceed 25 articles/second.

The code supporting NNTP reader clients is still under development. The addition of an article to a `name.map` file involves opening the file, reading its first few lines, then appending a single line to the end of the file. The disk I/O for these operations should compare quite favorably to an expensive file linking process. Updates for multiple articles which arrive shortly after the first in the same newsgroup may be consolidated into a single operation. Reading an article by number involves opening the `name.map` file, reading the first few lines, seeking directly to the corresponding entry, reading a line's worth of data, seeking to an offset within a cyclic buffer, and one (or more) data reads. Access to subsequent articles in the same group does not require re-reading the head of the map file. Article expiration on a cyclic INN server involves rewriting the `name.map` file, removing unwanted lines in the process.

Future Work

Because the cyclic buffer code is still experimental, a number of changes and additions remain to be done. A partial list, in no particular order, includes the following:

1. Incorporate the cyclic buffer code into the official INN source distribution, rewrite outstandingly-ugly hacks and "temporary" code as needed, and debug all components thoroughly.
2. Write a cyclic buffer library conforming to the INN soon-to-be-released "storage manager API." This programming interface will make multiple article storage methods (e.g., traditional spool, cyclic buffer, commercial database) much easier to write and maintain.
3. Pursue further improvements in the "history" database mechanism. The current DBZ implementation does an excellent job of putting a low upper-bound on the disk I/O required to query the database, but its average amount of disk I/O per query is too high. Servers that receive more than a few million article offers per day perform too much disk I/O in the current implementation.
4. Pursue further research into the NNTP reader newsgroup + article number => storage location mapping mechanism. In particular, replacing "overchan"'s current communication channel with INND would be a major improvement.
5. Add an option to make append-only buffers and allow INND to create such buffers as necessary. Some administrators wish to store articles for a guaranteed period of time. When articles stored within an append-only buffer are due to expire, the buffer file is removed in a single unlink() operation [Krten].

Concluding Observations

Before writing this paper, the author had never systematically measured the operating differences between a standard INN server and a cyclic one. The results surprised even him. Though the measurements and analysis presented is not comprehensive, it makes a compelling case for changing INN's article storage model. The implementation should be optimized further, and the NNTP reader client support requires debugging and much more real-world stress testing before this implementation can be considered for general use.

As the Internet continues to grow in popularity, the Usenet will certainly grow along with it, and that growth is not likely to drop far below current rates. Any reduction in "excessively-posted messages" will be offset by additional people using Usenet. The underlying store-and-forward mechanism does not scale well, ignoring problems with specific implementations: the current model of "broadcasting" articles to all Usenet servers is inefficient. Caching NNTP servers, such as [Assange], and automated article cancellation programs, such as [CM], are partial solutions at best. However, techniques such as IP multicast [Lidl], or World Wide Web-like centralized article storage are years away from acceptance. The Usenet will continue to require software optimizations and faster hardware until a better distribution mechanism is widely-implemented.

Availability

Cyclic buffer INN was first used at MRNet in July 1996 and development has proceeded, with fits and starts, since then. By the time this paper is published, the cyclic buffer code may already be merged into the official INN source distribution, albeit in a development branch of the source tree. Until then, the code is available via anonymous FTP under INN's original licensing restrictions (which are quite liberal) at ftp://ftp.mr.net/pub/fritchie/cnfs/. Also available in

that directory is the SE script used to monitor the servers' utilization stats and the raw data it generated. A mailing list for CNFS discussion is available by sending "subscribe cnfs" to <majordomo@ mr.net>.

The official INN source distribution is available at ftp://ftp.isc.org/isc/inn/.

Acknowledgments

This paper and the work it describes would not have been possible without the support and encouragement of MRNet Engineering Department staff, not to mention the typing, which saved a lot of wear and tear on sore hands. This paper would still be only partially-fired synapses if Lee Damon hadn't mentioned that the LISA '97 abstract submission deadline had been extended. Many people have assisted with the cyclic ideas and prototype implementations, from stress-testing to bug-fixing: Jerry Aguirre, Michael Beckmann, Barry Bouwsma, James Brister, Matt Bush, Evan Champion, Mark Delany, Avi Freedman, Darrell Fuhriman, Jeff Garzik, Joe Greco, Dave Hayes, Chris Halverson, John Ladwig, Clayton O'Neill, Alexis Rosen, Rich Salz, Robert "RS" Seastrom, Sang-yong Suh, Brad Templeton, Jeff Weisberg, Sven-Ove Westberg, and many others unintentionally omitted. Mike Horwath provided the reference INN server, since MRNet no longer has a standard INN machine. Special thanks to Dave Diehl, Olaf Hall-Holt, Andy Mickel, Joe St. Sauver, Peter Seebach, and John Sellens for their manuscript reviews. A case of Leinenkugel's goes to Nick Christenson for his extraordinary critiques. Finally, to the Norge '96 crowd, Louise Lystig Fritchie in particular, thank you for your support in this continuing madness called Life.

Author Information

Scott Lystig Fritchie graduated with a degree in mathematics and a concentration in computer sciecne from St. Olaf College, home of Minnesota's first UNIX machine and of "stolaf," the state's one-time gateway to the rest of the Usenet. Scott's other claim to (obscure) fame is implementing one of the world's first World Wide Web interfaces for a library cataloging system; it has since mutated into WebPALS, used by the PALS Across Georgia project and the Minnesota State Colleges and Universities (MnSCU). He currently works as senior systems administrator at the Minnesota Regional Network (MRNet) and can be contacted via email at <fritchie@mr.net>.

Bibliography

[Aguirre] J. Aguirre. Posting to news.software.nntp: "File by message ID instead of group/number." Message-ID <4gl99l$blm@olivea.ATC.Olivetti.Com>, Feb 23, 1996.

[Assange] J. Assange and L. Bowker. NNTPcache. ftp://suburbia.net/pub/nntpcache/.

[Christenson] N. Christenson, D. Beckemeyer, and T. Baker. "A Scalable News Architecture on a Single Spool." In *;login:*, Vol. 22, No. 3, June, 1997.

[CM] Cancelmoose. NoCeM. http://www.cm.org/.

[Cockcroft] A. Cockcroft. The SE Performance ToolKit. http://www.sun.com/960301/columns/adrian/.

[Collyer] G. Collyer and H. Spencer. "News Need Not Be Slow." In *Proceedings of the Winter 1987 USENIX Technical Conference,* Washington, DC, January, 1987.

[DejaNews] DejaNews. http://www.dejanews.com/.

[Delany] M. Delany. Posting to news.software.nntp: "cyclic news file system – more performance results." Message-ID <4lqh5o$kd6@bushwire.mira.net.au>, April 26, 1996.

[Dillon] Diablo: a backbone news transit system. http://www.backplane.com/diablo/.

[Graham] J. Graham. *Solaris 2.X: Internals & Architecture.* McGraw-Hill, 1995.

[Highwind] Cyclone NewsRouter. http://www.highwind.com/.

[Hitz] D. Hitz, J. Lau, and M. Malcolm. "File System Design For an NFS File Server Appliance." In *Proceedings of the USENIX Winter 1994 Technical Conference,* San Francisco, CA, 1994.

[Kantor] B. Kantor and P. Lapsley. "Network News Transfer Protocol." RFC 977, U. C. San Diego and U. C. Berkeley, February, 1986.

[Krten] R. Krten. "Improving Usenet News Performance." *Dr. Dobb's Journal,* May, 1996. See also: http://www.parse.com/.

[Leffler] S. Leffler, M. McKusick, M. Karels, and J. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley, 1989.

[Lidl] K. Lidl, J. Osborne, and J. Malcolm. "Drinking from the Firehose: Multicast USENET News." In *Proceedings of the USENIX Winter 1994 Technical Conference,* San Francisco, CA, 1994.

[McKusick] M. McKusick, W. Joy, S. Leffler, and R. Fabry. "A Fast File System for UNIX." *ACM Transactions on Computer Systems*, Vol. 2, No. 3, August, 1984.

[Poskanzer] J. Poskanzer. Multi-threading nnrpd and caching proxy. http://www.acme.com/java/software/Package-Acme.Nnrpd.html.

[Rakitzis] B. Rakitzis and A. Watson. "Accelerated Performance for Large Directories." *Technical Report 3006*, Network Appliance, Inc.

[Rivest] R. Rivest, "MD5 Digest Algorithm." RFC 1321, MIT and RSA Data Security, Inc., April 1992.

[StSauver] J. St. Sauver. "The 1996/97 Oregon Christmas USENIX Newsadmin Newsletter." http://darkwing.uoregon.edu/~joe/pdf/.

[Salz] R. Salz. "InterNetNews: Usenet Transport for Internet Sites." In *Proceedings of the USENIX Summer 1992 Technical Conference*, San Antonio, TX, June, 1992.

[Sedore] C. Sedore and E. Sedore. NNTPRelay Usenet News propagator. ftp://ftp.maxwell.syr.edu/nntprelay/.

[Swartz93] K. Swartz. "Forecasting Disk Resource Requirements for a Usenet Server." *Proceedings to the Seventh USENIX Systems Administration (LISA VII) Conference,* Monterey, CA, November, 1993.

[Swartz96] K. Swartz. "The Brave Little Toaster Meets Usenet." *Proceedings to the Tenth USENIX Systems Administration (LISA X) Conference*, Chicago, IL, September, 1996.

[Sweeney] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. "Scalability in the XFS File System." In *Proceedings of the USENIX 1996 Annual Technical Conference*, San Diego, CA, January, 1996.

[Tanenbaum] A. Tanenbaum. *Operating Systems: Design and Implementation.* Prentice-Hall, 1987.

[Templeton96a] B. Templeton. Posting to news.software.nntp: "How to do a better USENET file system." Message-ID <DnCo3D.Cz3@clarinet.com>, Feb 25, 1996.

[Templeton96b] B. Templeton. Posting to news.software.nntp: "Re: How to do a better USENET file system." Message-ID <DnFDJM.CMw@clarinet.com>, Feb 27, 1996.

[Top1000] Top 1000 Usenet sites survey. http://www.freenix.fr/top1000/.

Adaptive Locks For Frequently Scheduled Tasks With Unpredictable Runtimes

Mark Burgess & Demosthenes Skipitaris – Oslo College

ABSTRACT

We present a form of discretionary lock which is designed to render unreliable but frequently scheduled scripts or programs predictable even when the execution time of locked operations may grow and exceed their expected scheduling interval. We implement our locking policy with lock-unlock semantics and test them on the system administration language cfengine. The locks are controlled by too-soon and too-late parameters so that execution times can be controlled within fixed bounds even when scheduling requests occur randomly in addition to the periodic scheduling time. This has the added bonus of providing an anti-spamming functionality.

Introduction

When two or more instantiations of a program use a resource concurrently, it can lead to contention between the competing processes and unpredictable results. Application programs (for example, mail-readers or daemons) usually avoid this situation with the help of a lock so that only a single instantiation can run at a given time. A lock is a device which assures that only one process can use a specific resource at a given time [1, 2]; an application lock makes the execution of an application program into an exclusive resource. A typical approach to locking is to write a short file which contains the process ID of the running application [3]. The file has a unique name and is therefore trivially-locatable by multiple instantiations of the program. An application lock is generally adequate for programs which start and then expect to continue more or less indefinitely, e.g., daemons such as `cron` and mail readers, but it can lead to unfortunate problems if used to block frequently or periodically run programs or scripts. If the duration of such a program is capable of exceeding its scheduling interval, then there could be an overlap between instantiations of the program or a failure of the program to be started at the correct time, and this must be dealt with in a sensible way. This problem is of particular interest in connection with automated system administration where complex scripts are often scheduled by cron, but may also be started by hand.

Let us give a concrete example. Consider a program, run hourly by cron, which executes a remote command on a series of hosts; one can imagine a program which distributes or makes copies of key files. The time for this job to complete depends on many factors: the size of the files, the speed of the network, the load on the participating hosts etc. If the network latency time were high, or if an RPC error occurred then this script or program could hang completely or fail to complete inside its allotted hour. After the next elapsed hour, the hanging lock would result in an annoying error and a failure of the program to perform its task. If left unlocked, there could be contention between multiple instantiations of the program and inconsistent results.

Network services present a related problem. Consider a program which is initiated by a network connection to a particular port for the purposes of updating one or more resources. It is desirable to lock such a program to avoid contention between multiple connections. It might be appropriate to lock the entire process with a single threaded connection. Service demultiplexers like `inetd` contain the functionality required to serialize access. On the other hand, it might be better to lock only specific resources. We might wish to go even further and restrict the frequency at which the program can be run at all. Such a contingency could be used to prevent spamming of the network connection or even the accidental wastage of CPU time. All of the above examples may be thought of in terms of resource sharing in a concurrent environment.

If we focus our attention more to the problem of resource control we gain a new perspective on the problem of multiple program instantiations. Rather than locking an entire program, we lock smaller parts which are independent. The idea here is that it is useful to use discretionary locks to control only specific resources required within a program [4, 5]. Such 'local' locks might allow a program to run partially, blocking collisions, but would admit access to the busy resources on a one-by-one basis. This might not always be desirable though: the scheme could result in awkward problems if the resources were critical to the operation of the program as a whole. The program might be forced to exit without performing its function at all and thus time spent executing the partial-program would only be CPU time wasted. Unlike locking of kernel resources, or database operations, the co-existence of multiple programs does not necessarily require us to preserve every operation and serialize

them, it is sometimes sufficient to ensure that only the most up-to-date instantiation is allowed to run at a given time. It could also be acceptable to have different instantiations of a program running concurrently, but in such a way that they did not interfere.

In spite of all the conditionals in the above, it is possible to address a large proportion of the cases encountered by system tasks. In this paper we are interested in the first case where it is meaningful to lock self-contained 'objects' within a larger program (these are usually referred to as *atoms* in related literature). Such a scheme is appropriate for many system administration scripts which bundle resource-independent operations. We aim to create a more intelligent approach to the locking problem, which is robust to unpredictable failures and which provides certain assurances against hanging processes or failure to execute. We do not exclude multiple instantiations of programs running in different threads, but instead try to ensure that they cooperate rather than contend. The granularity of the locking scheme has to be chosen carefully to achieve sensible and predictable behaviour and we take some time to describe the behaviour in detail.

The locking semantics we describe here are motivated by our desire to equip the system administration robot `cfengine` [6, 7, 8, 9] with a flexible but autonomous mechanism for avoiding contention and spamming in a distributed, multithreaded environment. By introducing our new locking policy, cfengine can function as an integrated front-end for cron and network-initiated scripts, effectively creating a single network-wide file for starting regular and intermittently scheduled programs which is protected against spamming attacks or accidental repetition. Although intended for cfengine, the locking policy we have arrived at is applicable to any situation where programs are scheduled on a time scale which is comparable to their runtime. They would be particularly useful as an addition to scripting languages such as Perl, Guile, Tcl and even Java.

Traditionally attention has been given in the literature to the problem of locking of shared memory resources and concurrent database transactions using discretionary locks, mutexes, semaphores and monitors [4, 5]. Resource locks in distributed systems have also been discussed in connection with fragile communication links [11, 12] and independent parallelism [13]. Application locks do not seem to have enjoyed the same interest in the literature, perhaps because of their apparent triviality, but they are widely used in concurrent and shared applications and are closely related to all of the above issues.

In the present work we wish to illustrate how a simple modification of the most trivial application locks can lead to enhanced autonomy of scheduled systems. By implementing such locks in the automated system administration robot cfengine we show how this is directly relevant to the reliability of automated system administration. Our locks are a generalization of the concurrent lock concept, ignoring the strong ordering of the atoms, but including garbage collection and protection against undesirable repetition.

Locking Semantics

All locking begins by defining *atomic operations*, or critical sections: these are the basic pieces of a program that must run to completion, without the disturbance from third parties. In other words, atoms are all-or-nothing pieces of a program. Atoms are protected by `GetLock()`, `ReleaseCurrentLock()` parentheses within the program code.

```
GetLock (parameters)
        /* Atom code */
ReleaseCurrentLock()
```

Serialized access to these atoms is assured by encapsulating each one with an exclusive lock. To create a locking policy, one must find the most efficient way of implementing resource control. If we lock objects which are too primitive (fine grain), we risk starting programs which will only run partially, unable to complete because of busy resources. This would simply constitute a waste of CPU time. On the other hand, if we lock objects which are too coarse, logically independent parts of the program will not be started at all. This is unnecessary and inefficient. In a concurrent environment there is no reason why independent atoms could not run in separate threads, allowing several instantiations of a batch program to 'flow through' one another. This assumes however that the order of the atoms is not important.

By defining suitable atoms to lock, one is able to optimize the execution of the tasks in a program. Several approaches to locking may be considered. A lock-manager daemon is one possibility. This is analogous to many network license daemons: a daemon hands out tickets which are valid for a certain lifetime. After the ticket expires, the program is considered overdue and should be killed. A major problem with a daemon based locking mechanism is that it is highly time consuming and that it is susceptible to precisely the same problems as those which cause the uncertainty on program runtimes. Use of `flock()` is another possibility, but this is not completely portable. A realistic approach needs to be more compact and efficient.

Implementation

We have chosen to implement locks using regular files and index nodes. By using a unique naming algorithm, we are able to instantly 'know' the name of a lock without having to search for it or ask a manager for the data. This minimizes time consuming calls to the network or to disk.

In order to secure a unique name, we need to provide enough information to be able to identify each atom uniquely. This is presently accomplished by passing a string to the lock function which can be combined with other elements such as the host on which the lock was created and any other relevant information. For convenience we classify atoms with an operator/operand pair. For example, consider a lock request to edit a file. In this case the operator would be "edit" and the operand would be the name of the file. The names must be processed to expunge unfortunate characters which would lead to illegal file names.

```
CanonifyName(char *buffer)
{
for (sp = buffer;
        *sp != '\0'; sp++)
   {
   if (!isalnum(*sp))
      {
      *sp = '_';
      }
   }
}
```

We use a function `CanonifyName(`*name*`)` which returns a string suitable for use as a filename. It suffices to swap illegal characters for an underscore, for instance.

In order to function properly, the lock-name must be different for each distinct atom, but must be constant over time so that multiple instantiations of the program will always find the same lock. The time information should therefore not be coded into the name of the lock; instead, one relies on the time stamps on the inodes to determine their age. For example, when editing the file `/etc/motd` on host `dax`, a lock named

```
lock.cfengine_conf.dax.
                editfile._etc_motd
```

would be created.

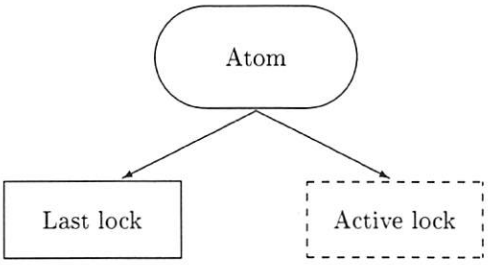

Figure 1: The adaptive lock components for an atom.

We create two kinds of lock within the `GetLock()` call: a lock for active threads of execution which blocks multiple instantiations of a process, and a permanent lock which records the last time at which the resource was accessed; see Figure 1. The latter information can be encapsulated in a single inode without using any disk blocks and provides the information necessary to restrict the frequency of access. We call this an anti-spamming lock.

If a lock already exists for a specified atom, and that lock has not expired, the atom remains locked and access to the atom is denied. Lock expiry occurs when a certain predefined period of time has elapsed since the active lock was created. In this case, a garbage collection mechanism attempts to carefully eliminate the process attached to the hanging lock (if it still exists)

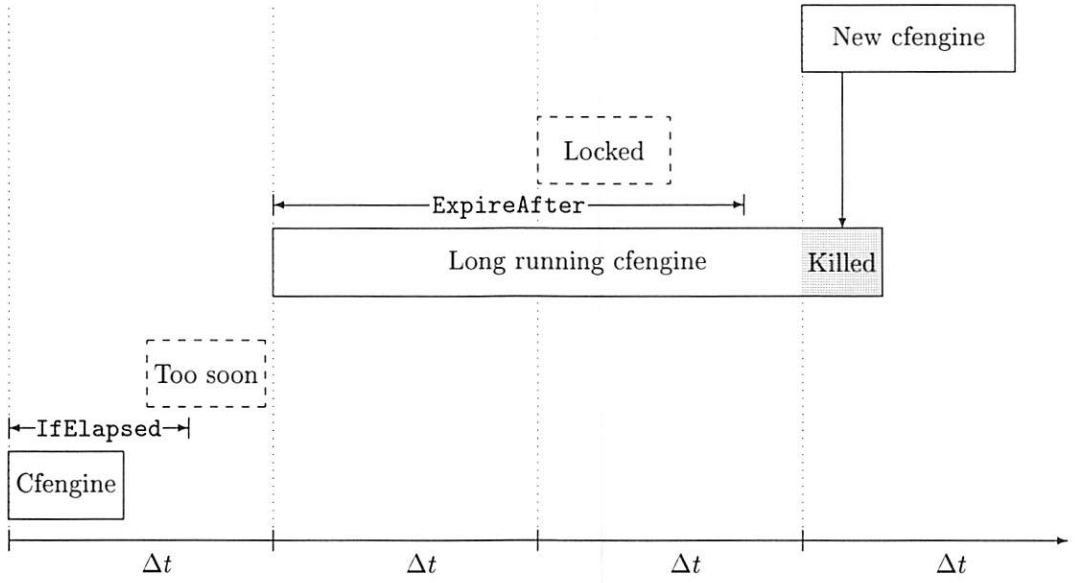

Figure 2: A schematic illustration of the behaviour of locks with respect to the scheduling interval Δt and the parameters `IfElapsed` and `ExpireAfter`.

and then remove the old lock, replacing it with a new one and permitting the killing process to take over the task. The third possibility is that no active lock exists for an atom, but that the time since its previous execution is too short. This information is gleaned from the permanent lock. In that case access to the atom is also denied. This feature gives us the 'anti-spamming' functionality. A record of these lock transactions is kept for subsequent analysis if required. This indicates when and how locks were created and removed, thereby allowing problem cases, such as locks which always need to be removed forcibly, to be traced.

To make the locking behaviour user-configurable we introduce two parameters called `ExpireAfter` and `IfElapsed`, which have values in minutes. See Figure 3. `ExpireAfter` describes the number of minutes after creation at which a lock should expire. It is measured from the creation time-stamp on the active lock to the current value of the system clock. The variable `IfElapsed` describes the number of minutes after which it becomes acceptable to execute the same atom again. It is measured from the modification time stamp of the anti-spamming lock to the current value of the system clock.

We choose to read the current time as a parameter to `GetLock()`, rather than reading it directly in the locking function, for the following reason. The most correct time to use here could be construed in one of two ways: it could be taken as being the time at which the program was started, or as the exact time at which the lock creation takes place. The difference between these times could differ by seconds, minutes or hours depending on the nature of the job being locked. By using the time at which the program was started for all locks throughout the program, one

```
GetLock(operator,operand,ifelapsed,expireafter,host,now)
{
sprintf(LOG,"%s/program.%s.runlog",LOGDIR,host);
sprintf(LOCK,"%s/lock.%s.%s.%s",LOCKDIR,host,operator,operand);
sprintf(LAST,"%s/last.%s.%s.%s",LOCKDIR,host,operator,operand);

lastcompleted = GetLastLock();      /* Check for non-existent process */
elapsedtime = (now-lastcompleted) / 60;

if (elapsedtime < ifelapsed)
   {
   return false;
   }

lastcompleted = CheckOldLock();     /* Check for existing process */
elapsedtime = (now-lastcompleted) / 60;

if (lastcompleted != 0)
   {
   if (elapsedtime >= expireafter)
      {
      pid = GetLockPid();           /* Lock expired */
      KillCarefully(pid);
      unlink(LOCK);
      }
   else
      {
      return false;                 /* Already running */
      }
  }

SetLock();
return true;
}
```

Figure 3: A schematic algorithm for implementing the locking policy. The function call `GetLock()` takes arguments which are used to build a unique name. Operator and operand pertain to the atom which is to be locked. The expiry time and elapsed time limits are times in minutes, and the `now` parameter is the system clock value for the time at which the lock is created. The function `GetLastLock()` creates the anti-spamming 'last' lock if it does not previously exist. This is important for theoretical deadlock avoidance.

effectively treats a 'pass' of the program as a cohesive entity: if one lock expires for a given value of `ExpireAfter`, they all expire. A certain ordering of atoms can be preserved. If, on the other hand, one always reads the present value of the system clock directly, the locking mechanism becomes sensitive to the length of time it has taken to execute the different parts of the program. Both policies might be desirable in different situations, so we do not see fit to impose any particular restriction on this.

When a lock has expired, we try to kill the owner process of the expired lock. The process ID of the expired process is read from the active lock. Then the signals `CONT`, `INT`, `TERM` and `KILL` are sent in that order. On some systems, `INT` is the only signal that will not hang the process permanently in case of a disk-wait situation, thus `INT` is sent first. Then the default terminate signal `TERM` is sent, and finally the non-ignorable signal `KILL` is sent. Sleep periods of several seconds separate these calls to give the kernel and program time to respond to the signals. The `CONT` signal is placed first in case the process has been suspended and wants to exit straight away. This should be harmless to non-suspended processes.

Adaptive Locks In cfengine

Our locking mechanism was designed and implemented with cfengine version 1.4.*x* in mind. Cfengine is a descriptive language and a configuration robot which can perform distributed system administration on large networks [6, 7, 8, 9]. A cfengine program is generally scheduled as a cron job, but can also be initiated interactively or by remote network connection. Cfengine can examine many hundreds of files, system processes and launch dozens of user scripts depending on the time of day and the host concerned. Cfengine's job is to coordinate these activities based the *state* of the system. The state comprises many variables based on host type, date, time, and the present condition of the host as compared to a reference model. Its total run time involves too many variables to be practically predictable.

Opening cfengine to the network places an extra onus on its behaviour with respect to scheduling. Although designed in such as way that it does not give away any rights to outside users, cfengine is intentionally constructed so that general users (not just `root`) can be allowed to execute the standard configuration in order to update or diagnose the system, even when human administrators are not available. The mere thought of this is enough to send convulsions down the spines of many system folks, and it would indeed be a cause for concern unless measures were incorporated to protect such a provision from abuse. Adaptive locks will therefore play a central role in a 'connected' cfengine environment in the future.

We have tested our adaptive locks with cron initiated cfengine as well as with remote connections with some success. The locks do indeed fulfill their role in preventing seizures and overlaps which can occur due to unforeseen delays. The locking of individual atoms means that, even though a particular script might run over its allotted time, other scripts and tasks can be completed without delay, come the next scheduling time. Silly mistakes can also be dealt with unproblematically: a cfengine program which starts itself is impervious to the apparent recursive well, provided the `IfElapsed` parameter is not set to zero. This is, after all, simply an example of spamming (see below).

Adaptive locks are very important for cfengine: cfengine is a tool which is supposed to automate basic system administration tasks and work as a front end for user-scripts, allowing administrators to collect an entire network's scripts into a single place and providing a net-wide front-end for cron. In order to be effective in this role, cfengine must support a high degree of autonomy. Cfengine atomizes operations in different ways. Some operations, such as file editing and script execution, are locked on a per-file basis. Other operations which could involve large scale traversals of the file system are locked per class of operation. The aim of the locking policy is to make the system safe and efficient – i.e., not to overload to the system with contrary tasks.

Previously, cfengine processes were locked by a single global lock. If a process were interrupted for some reason, a hanging lock would remain and cause warning messages to be printed from the affected hosts the next time cfengine was scheduled. Certain cfengine processes would overrun their allotted time: typically the weekly runs which perform extensive system checking and updates of system databases. This would happen once a week, generating useless mail which everyone would have been happier not to receive. Bad NFS connections through buggy kernels have been known to hang scripts. Also, bugs in cfengine itself, which manifest themselves only under special conditions, could result in a core dump and a hanging lock. Although each isolated occurrence of these problems was relatively rare, the cumulative effect on a large network could be substantial enough to be an irritation. The system administrator would then be required to chase after these old locks and remove them.

The new locks allow several cfengines to coexist as different processes, without interference. Moreover, since one of the purposes of automation is to minimize the amount of fruitless messages from the system, the original locking policy was clearly not in tune with the cfengine's autonomous philosophy. Using the new adaptive locks, cfengine can clean up its own hanging processes without the intervention of a human, and even better: silently. In large network environments such silence is golden.

With large scale system checking, the total number of locks used in a single pass of cfengine might approach several tens or even a hundred on an busy system, but only one active lock is present per active thread. (We do not normally expect more than two threads for normal system administration tasks.) The anti-spamming locks take up only a single inode each and since most file systems have thousands of spare inodes, this usage is hardly a concern. The first part of Table 1 shows runtimes for a small cfengine run which sets 24 locks, while the last part shows a run which sets 32 locks. Some of the operations involved in the second run are large. Although the difference in real time seems large for the smaller run, the difference in user and system time is much smaller. The actual CPU time spent to set and remove the locks is not high, which means that we wait for the disk when creating and deleting the locks. For the larger run, the differences are almost the same, but here the dominating part of the run is the cfengine operations itself, not the administration of locks.

Cfengine can be exposed to infinite loops from which it will recover gracefully. Figure 3 illustrates a cfengine program which calls itself. Suppose we have a cfengine program which contains three atomic operations *A*, *B* and *C*. Suppose also that *B* is a shell command which executes cfengine. Let us then examine how the locks handle the execution of this program, assuming i) that the scripts have not been executed for a long time > `IfElapsed` and ii) that the locking parameters have 'sensible' values.

Small	locks	no locks	diff
real	3.3	1.1	2.2
user	0.3	0.3	0
sys	0.4	0.3	0.1

Large	locks	no locks	diff
real	12.8	10.0	2.8
user	2.3	2.2	0.1
sys	4.2	4.0	0.2

Table 1: Real, user and system time in seconds for two different cfengine runs.

The example in Figure 4 shows how no more than two cfengine processes will be started. When cfengine is first started, it executes atom *A*, locking and unlocking it normally. When it arrives at *B*, a lock is acquired to run cfengine recursively (since this has not previously occurred) and a second cfengine process proceeds to run. Under the auspices of this second process, a new lock is requested for *A*, but this fails since it is too soon since the last instance of *A* from process #1. Next a lock is requested for *B*, but this also fails because *B* is busy and not enough time has passed for it to expire. Thus we come to *C*. Since *C* has not been executed, cfengine obtains a lock for *C* and executes it to completion, then releasing the lock. Process #2 is then complete and so is atom *B* from cfengine process #1. The lock for *B* is released and cfengine attempts to finish process #1 by getting a lock for *C*. This fails however, since *C* was just

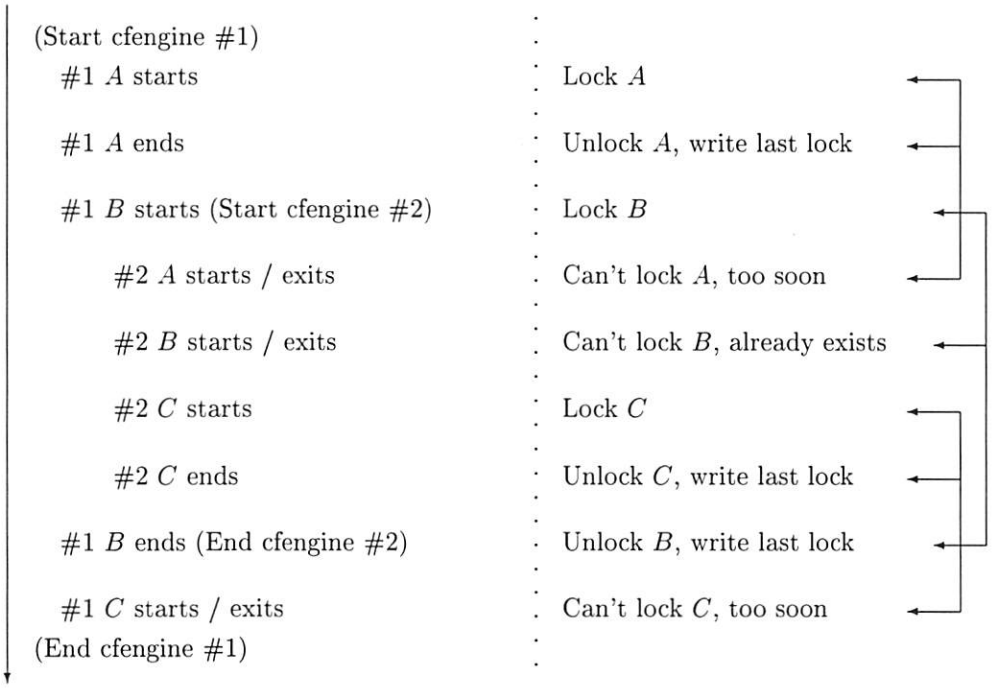

Figure 4: Diagram of actions versus time for a cfengine process which calls itself recursively. This illustrates the way the locks prevent infinite recursive loops.

executed by the process #2 and not enough time has elapsed for it to be restarted (or killed). The first process is then complete.

Notice how two processes flow through one another. The real work in *A* and *C* (which could have been done by a single process) simply gets shared between two processes, and no harm is done.

A similar sequence of events occurs if a process hangs while executing an atom (see Figure 5). Suppose that an old instantiation of (process #1) managed to execute *A* successfully, but hung while executing atom *B*. Later, after the lock on *B* has expired, another cfengine (process #2) will execute *A* again, kill the previous lock on *B* and execute *B*, then execute *C*. Here we assume that *B* hangs for some spurious reason, not because of any fundamental problem with *B*.

Similar scenarios can be constructed with remote connections and more convoluted loops. All of these either reduce to the examples above or are defeated by cfengine's refusal to copy from a host to itself via the network (local copying without socket waits is used instead). Spamming attacks from malicious users are stifled by the same anti-spamming locks.

For various reasons our implementation of locks in cfengine includes logging of lock behaviour. This allows us to trace the executing of scripts and other atoms in a cfengine program and gain an impression of how long the individual elements took to complete. This information could then be fed back into the locking mechanism to optimize the parameters `IfElapsed` and `ExpireAfter`. We have also added a parameter to limit the maximum number of cfengine processes which may be started simultaneously to cover various contingencies whereby multiple cfengines might be started unintentionally. One example of this is that a hanging NFS filesystem might hang cfengine over a long period, preventing it even from receiving expiry signals which would normally clear up the problem.

Deadlock and Strange Loops

One of the drawbacks with locking mechanisms is that they can, through unfortunate interactions, lead to deadlock if the system in which they are used admits circular dependencies. In most of the cases we encounter in system administration the likelihood of this occurring is insignificant, but there is nonetheless a theoretical possibility which is worth addressing.

In a single threaded application, the use of our locking policy renders deadlock impossible unless the `ExpireAfter` or `IfElapsed` parameters are set to silly values (see next section). Deadlock (a non-recoverable hang) can only occur if concurrent processes are running in such a way that there is circular waiting, or a tail-chasing loop. We assume that the atoms themselves are safe, since no locking policy can protect completely against what happens inside an atom.

Starvation is a possibility however. This means that certain actions may not be carried out at all. A simple of example of this is the following: if every instance of an atom overruns its allotted time, and the expiry time for the atom is shorter than its scheduling interval, then the atom will be killed at every scheduling interval, never completing its task even once.

A related concern is that of spamming, or the senseless repetition of a given atom, either by accident or through malice. Adequate protection from spamming is only assured if the `IfElapsed` parameter is not set to a very low value.

An atomic operation which contains an implicit call to itself will never be able to enter into an infinite loop, since the locks prevent more than a single instantiation of the atom from existing.

Predictable Behaviour

The success of any locking policy depends on the security of the locks. Locks must be impervious to careless or malicious interference from other processes or users. If not secure from interference, it is a

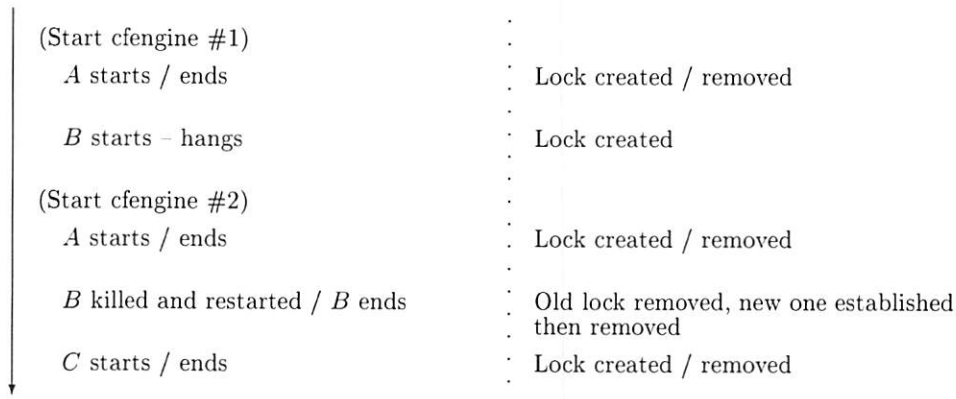

Figure 5: Diagram of actions versus time for a cfengine process which has hung while executing some action *B*. The lock expires and a new cfengine takes over the remaining work, killing the old process along the way.

trivial matter to defeat the locks and open programs to unpredictable behaviour. Deleting all the lock inodes suffices to subvert the locking mechanism.

Locks may also be defeated by setting the parameters `ExpireAfter` and `IfElapsed` to zero. In the former case, proper exclusion of contentious processes will be disabled, and in the latter protection from spamming will be disabled.

`IfElapsed`	Result
T	Prevents atom from executing too soon (before T minutes)
$T=0$	Impotent, spamming possible
$T \to \infty$	Run atoms once only

Table 2: Lock behaviour for various values of the variable `IfElapsed`

It is assumed that users of the locks will not sabotage themselves by setting these parameters with silly values. It would be an interesting investigation to see whether optimal values of these parameters could be found for a specific type of atom, and whether the values could even be determined automatically.

`ExpireAfter`	Result
T	Makes atoms interruptable after T minutes
$T=0$	Impotent, nothing is locked
$T \to \infty$	Never expire, deadlock possible

Table 3: Lock behaviour for various values of the variable `ExpireAfter`

In order to deal with atoms which frequently overrun their allotted time, we may note a rule of thumb, namely that `ExpireAfter` should generally be greater than `IfElapsed`. If `ExpireAfter` < `IfElapsed`, expiry will occur every time a new atom is started after an overrun. This is probably too soon, since the aim is to give the atoms a chance to complete their work.

The function of the active locks is to enforce a correct or sensible interleaving of the atomic operations. The optimal definition of atoms can play a key role in determining the correctness of behaviour. The so-called locking *granularity* is central to this issue. A central assumption in our treatment here is that the atoms themselves do not lead to subversive or incorrect behaviour. No locking policy can effectively restrict what happens within the atoms.

To illustrate the importance of granularity, consider an extreme example in which two concurrent threads contain a circular wait loop. Suppose two threads each run at regular intervals (Figure 6).

Thread #1 performs two operations in sequence: the first is to sleep until object X is created, the second is to create object Y. In thread #2, the operation sleeps until Y is created and then object X is created. Clearly neither thread can proceed in this circular wait loop and deadlock ensues.

Let us now consider the two alternative ways of locking these actions and the resulting behaviour. If we lock both actions as a single atom, then expiry will cause the threads to die and be restarted after a certain

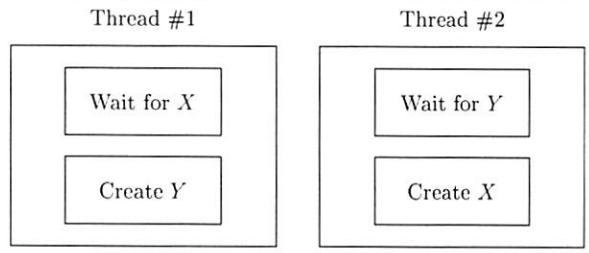

Figure 6: A two threaded example with circular waiting.

time. However, each time the threads are started, they fall into the same trap, since they can never proceed past the first operation. If, on the other hand, we lock each operation as separate atoms, the deadlock can be broken *provided the locks are correctly removed from the killed process and the anti-spamming locks are updated.*

Then the scenario is as follows: The first time the threads run, they fall into deadlock. After a certain time, however, the threads expire and one or more threads is killed when a new process tries to run the atom. If we assume that `IfElapsed` is greater than Δt, the scheduling interval of the program then, as the new threads start, insufficient time will have elapsed since the last lock was written for each thread, and the first operation will not be executed. This allows the offending atom to be hopped-over and the deadlock will be circumvented.

The assumptions in this scenario are clear:

- Locking each operation separately implies that it is safe to execute the operations independently.
- The `IfElapsed` parameter must be set to a value which is greater than scheduling time for the atom (not necessarily just its encapsulating program), and the anti-spamming lock must already exist. This means that the permanent lock should always be created if it does not already exist, otherwise deadlock is possible.
- The `ExpireAfter` parameter must be set to a value which is greater than the scheduling interval.

No greater assurances against deadlock can be given, nor do we attempt to cover every avenue of circular dependency. The possible cases are quite complicated. If silly values are chosen for the parameters

IfElapsed and ExpireAfter, we can theoretically end up in a deadlock situation. For our purpose of utilizing locks in autonomous system administration, the likelihood of such strange loops is small and of mainly theoretical interest. We therefore decline to analyze the problem further in this context, but end with the following claim. If Δt is the scheduling interval (the interval at which you expect to re-run atoms), then

$$\texttt{ExpireAfter} > \Delta t \geq \texttt{IfElapsed}$$

ensures correct and sensible behaviour. What ever one sets ExpireAfter to, its true value can never be less than that IfElapsed, since this defines the rate at which the locks are reexamined.

All of these theoretical diversions should not detract from the real intention of parameters: namely to provide reasonable protection from unforeseen conditions. For normal script execution, on an hourly basis, we recommend values approximately as follows:

Δt	1 hour
IfElapsed	15 mins
ExpireAfter	1 hour 30 mins

Conclusions

The locking policy introduced in this paper essentially solves the problem of hanging and crashed processes for cfengine, using a minimum of system resources. Although the simplicity of the algorithm could make the autonomous garbage collection procedure inappropriate for certain programs, in most cases of interest to system administrators, the behaviour is sensible and correct. The principal advantage of these locks is that one can always be confident that the system will not seize up; the flow of updates remains in motion.

How do system administrators use these locks in practice? The simplest way, which is completely transparent, is to use cfengine as a front-end for starting all scripts. The has several advantages, since cfengine provides a powerful classing engine which can be used to make a single net-wide cron file. Cfengine can do many things, but it is valuable even solely as a script scheduler. The alternative to this is to implement the locks in Perl or shell or some other scripting language. This is easily accomplished since the locks use only files (`echo >> file`) and inodes (`touch file`). Time comparisons are harder in the shell, but not insurmountable. Languages like Perl and Guile/scheme should implement the locks as a library module.

One minor problem we have run into occurs with programs which are started through calls to rsh. In this case, the rsh process does not always terminate, even when the process started by rsh has exited. If such a program is killed, when a lock expires, processes will not necessarily die in the intended fashion. Thus while the new instantiation of the program may continue to restart the entire task anew, this can leave hanging processes from the ostensibly-killed instantiation, which simply clutter up the process table. A possible solution would be to kill the entire process group for the rsh, but this method is not completely portable. This is presently a teething problem to be solved.

Adaptive locks contribute an insignificant amount of time to the total runtime in trials with cfengine and conceal the occurrence of spurious messages associated with the locks. Our locks are simple to implement and may be used in any program where one has atomic operations whose order need not be serialized into any strong order. An added side effect is that programs become effectively re-entrant to multiple threads.

It would make a fascinating study to determine whether the intelligence of a program like cfengine could be extended to encompass learning with respect to the jobs its carries out. Could, for instance, the values of IfElapsed and ExpireAfter be tuned automatically from the collective experience of the system itself? For example, an atom which is frequently killed could be allowed more time to complete. Conversely, programs which are started at every IfElapsed interval could indicate an attempt to spam the system, and measures could be taken to warn about or restrict the use of that atom. It is surprising how many interesting issues can be attached to such a simple idea as the adaptive lock and we hope to return to some of them in the future, as part of our programme of research into self-maintaining operating systems.

Availability

Source code for our locking scheme is available as part of the GNU cfengine software distribution. This may be collected from any GNU repository, or from the URL in [7]. All of this software is distributed under the GNU public license.

Author Information

Mark Burgess received a Ph.D. in theoretical physics in 1990 from the University of Newcastle Upon Tyne. Since then Mark has worked both in theoretical physics and computing science with complete disrespect for subject boundaries. He is the author of several books and of the system administration robot 'cfengine.' He is now associate professor at Oslo College and is considering submitting a Ph.D. in computing science, since it pays better. Mark can be contacted at <mark@iu.hioslo.no> and he is caught in some sort of web at http://www.iu.hioslo.no/~mark .

Demosthenes Skipitaris has a Master's degree in informatics from the University of Oslo, Norway. Demosthenes worked as system manager for the supercomputing installation at the University of Oslo for several years and has worked closely with

distributed filesystems like DFS. He is now with the Faculty of Engineering at Oslo College where he and Mark play Dr. Frankenstein with cfengine. His email address is <ds@iu.hioslo.no> and he is waiting to be eaten at http://www.iu.hioslo.no/~ds .

Bibliography

[1] Gregory V. Wilson, *Practical Parallel Programming*, MIT Press (1995).

[2] G. R. Andres, *Concurrent progamming*. Benjamin Cummings (1991).

[3] Typical applications which use this approach are the Unix `cron` daemon and the `elm` mail reader, to mention just two.

[4] A. S. Tanenbaum, *Distributed Operating Systems*, Prentice Hall, London (1995).

[5] O. Wolfson, "Locking Policies for Distributed Databases," *International Conference on Data Engineering*, 315 (1984).

[6] M. Burgess, "A site configuration engine," *Computing Systems*, **8**, volume 3, 309 (1995).

[7] The cfengine web site: http://www.iu.hioslo.no /mark/cfengine .

[8] M. Burgess and R. Ralston, "Distributed resource administration using cfengine," *Software Practice and Experience* (in press).

[9] Cfengine, documentation, Free Software Foundation 1995/6. This is also available online at the web site [7].

[10] O. Wolfson, "The Performance of Locking Protocols in Distributed Databases," *Proceedings of the Third International Conference on Data Engineering*, 256 (1987).

[11] A.P. Sheth, A. Singhal and A.T. Liu, *International Conference on Data Engineering*, 474 (1984).

[12] L. Chiu and M.T. Liu, *Proceedings of the Third International Conference on Data Engineering*, 322 (1987).

[13] M. Herlihy, "Wait Free Synchronization," *ACM Transactions on Programming Languages and Systems,* **13**, 124 (1991).

Creating a Network for Lucent Bell Labs Research South

Tom Limoncelli, Tom Reingold, Ravi Narayan, Ralph Loura
– Lucent Bell Labs

ABSTRACT

This paper describes the tools and techniques used to split the AT&T Bell Labs Research networks in Holmdel and Crawford Hill into separate networks for Lucent Bell Labs and AT&T Labs as part of the "tri-vestiture" of AT&T in 1996. The environment did not permit us to keep the system down for an extended period of time. Legacy systems and old configurations were supported while new systems were introduced. We did not have direct control over many machines on our network. This paper describes the old network and what we were trying to build (or split), but focuses mostly on the specific techniques we used or developed. What made our network unique is the amount of self-administered machines and networks in our environment. We took unmanaged chaos and created two clean networks. This paper is from the perspective of the Lucent Bell Labs system administrators (SAs), not the AT&T Labs SAs. The transition did not go smoothly, and if we could have read this paper before we began we would have avoided many of the problems.

The beauty of it all is that we did not take one mess and create two separate ones, we split and cleansed the networks at the same time.

The Increasing Trends

There is little literature [RFC1916] on the topic of merging, splitting, and renumbering computer networks because when it is done it is often not thought worthy of documenting. We initially did not plan to document our overhaul because we felt we were "the only ones"[1]. However soon we realized this was not the case for many reasons. [RFC1900] [RFC2071] [Lear1996].

Old networks need to be cleaned. We see a growing trend in research and academic environments where networks have grown organically and ultimately reach a point where they must be pruned and weeded. Networks that grow this way often do so because they are renegades that exist outside of any authoritarian central control. In recent years corporate CIO departments have down-sized from mainframes to Unix environments. Now "the suits" use the same computers and protocols that were once the domain of the renegades. Soon centralization or at least standardization becomes an obvious way to improve performance, stability, and, of course, to save money. It can also happen for political reasons. These situations can all trigger the merging of networks.

The Reagan Era was marked by constant corporate buyouts. Now some monolithic companies are splitting themselves. We feel that corporate divestitures such as AT&T's may be a growing trend. The Federal Trade Commission is currently investigating at least one large software company.

[1] at least an insignificant minority.

It was tempting to take one messy network and create two messy networks. We avoided the temptation and instead viewed the corporate split as an opportunity to base-line our systems and generate two clean, well-engineered, networks.

The Old Way

From informal surveys, we found that most sites perform network merges and splits in very simple and unsophisticated ways:

1. Declare a week, weekend, or evening to be "downtime" and convert every machine at once.
2. Move one machine at a time.

We preferred the first option, but management wouldn't approve a simultaneous vacation of approximately 500 researchers. Obviously the first option could have been a disaster if we made the same change to every machine over a weekend and then discovered (presumably on Monday morning) that the changes we made were wrong. The second option was too labor intensive. It involves a lot of footwork as a personal appointment must be made with each user. In a chaotic environment keeping appointments can be difficult.

We adopted a hybrid technique.

The Task

On September 20, 1995, AT&T announced that it would separate into three companies: AT&T, which would retain the Long Distance and other Services businesses; Lucent, which would retain the telephony

and data systems and technology business; and NCR, which would retain the computer business that they had before they were acquired. Our user base and network had consisted of the computers and networks related to AT&T Bell Labs Research in Holmdel, NJ and the Crawford Hill Laboratory, also in Holmdel, NJ. Thanks to fiber optics,[2] the networks were tightly coupled even though they were three miles apart.

Unlike some divisions that were targeted entirely to one company or another, we were split approximately 40% for AT&T, the new "AT&T Labs," and 60% for Lucent, the new "Bell Labs." None of our users was targeted for NCR.

Before the split was announced we had planned massive changes. Luckily we hadn't yet deployed them. Our plan was to take our 40-odd haphazardly grown networks (consisting of 600 computers) and replace them with one large, switched ethernet network per building, with a number of small subnets for special purpose work.

With the announcement of the company split, we had to adapt our plans to the new scenerio. Luckily we had already done the homework required. We essentially continued with our plan, but did it once for each company. That is, two major switched ethernet networks per building (one for each company) and a dozen or so small subnets for special purpose work.

We only had to split our networks to the point where we had two, completely independent connections to the corporate backbones. AT&T and Lucent would deal with splitting the backbone. They would even relay traffic for approximately six months, though we could use filters on our own routers to simulate the final split to "check our work."

Why Our Network Had Grown Organically

Our networks were as organized as a swamp. Originally each small department had self-administered its networks. Departments had been reorganized, system administration had been centralized, re-centralized, gained management support and lost it a couple of times. Many users maintained their own machines for historical reasons. In some groups, the SAs were considered "secretarial" and were only expected to create accounts, hand out IP addresses and do backups. The combined network we inherited was therefore quite chaotic. For example, when a researcher felt the network was too overloaded, he would add a second ethernet port to his workstation and create a small subnet for himself. This worked because we used RIP at the time. One of the SA teams even kept /usr/local world-writable so users could install new software as they wanted!

[2]Key elements of fiber optics were invented at the Crawford Hill lab.

What Our Network Looked Like

The network consisted of four main user communities each with its own SA procedures and standards. Although we had recently centralized SA functions, the four communities had never been properly merged. But at least by then, we had eliminated 90% of the UID conflicts. There were four NIS domains, two different automount configurations, and each group had a different /usr/local. Each community had from two to five main networks plus many subnets for experiments or to pacify researchers that felt they were important enough to warrant private networks. As a result, we had approximately 40 Class C-sized networks. We had recently been allocated a Class B network which we had planned on dividing into subnets and transitioning to but this migration had only just begun. Almost all of our machines were connected via 10base-T wiring which was administered by our telecom department. There were also 2-3 pockets of thinnet which were maintained by the local users and were being replaced by telecom's 10base-T wiring as part of this project.

Before the split was announced, we wanted to build 2 major networks for Bell Labs, one in each building, with small subnets for experiments only. We wanted no more private networks than needed. We wanted all networks to be connected to our routers, not hung off a Unix host with two ethernet interfaces. The major networks would consist of ethernet switches tied together with FDDI or some other high-speed network technology. We wanted one NIS domain, one /usr/local (or equivalent), one procedure for everything. When the split was announced, we took our plans and doubled everything: one for each company. We would move machines to the correct NIS domain, network, cut the networks, and be done.

NIS Domains and Stranger Things

Changing the IP address of a machine affects all the machines that depend on it for services (i.e., the clients have to be rebooted after the server is changed). NIS and DNS have IP addresses hardcoded making the change even more difficult. Rather than merge three of the NIS domains into an existing fourth, we created a new NIS domain and merged all four into this new domain. We had to reconfigure more clients this way, but it permitted us to start with a clean NIS configuration that we could experiment with, install security-enhancement tools, and develop modern update procedures and more. It also let us start with a fast machine. Since upgrading an NIS master server is difficult, it's best to use the fastest machine you have on hand, deferring the next difficult upgrade. We were also unsure of what patches and security problems [Hess1992] might have existed on the old master. ([Wietse1996] and other tools help make make NIS more secure.)

Most NIS masters read their source files directly from where Unix usually puts them (i.e., passwd is in /etc). We instead put all NIS databases in /var/yp/master (which we aliased to $M) and used `xed` to edit the files. The advantages are:

- $M/passwd contains the global passwd file while /etc/passwd only contains a minimal password file. Break-ins that steal the NIS passwd table are useless for breaking into the master.
- The /etc/passwd file on the master is very small. It contains only the accounts required (system administrators) and the passwords for these accounts are different than the ones in the global NIS passwd table.
- There is no need for `root` (or any UID 0 account) to be in the NIS passwd table. All clients retain a local root password. If the NIS passwd table is stolen and the root password is cracked, the intruder does not have the keys to all the workstations.

"`xed`" is a script that automates file locking and RCS (to record revision history), and then runs $EDITOR to edit the file. After any file is edited, "`ypmake`" (our own script) is run manually to perform 100% of the update processing. "`ypmake`" locks a file and runs the Makefile. The benefits are:

- Can't step on people's toes: Multiple people editing the same file or running "make" at the same time had caused problems in the past.
- Reduced training time and confusion: The old systems had each administrative file on a different directory on possibly a different machine. Training consisted more of "where is this file and what do I do after I edit it" than what we consider "real work." Now, no matter what you needed to change the procedure was "`xed $M/foo`" followed by "`ypmake`."
- Since `xed` maintains a RCS log of all changes, we can revert to an old revision of any file at any time. This has proved useful when an error crept into a file, we could use the RCS log to discover when the mistake was made (how long was that service misconfigured) and who made the mistake (so they can be re-educated).

Deciding what to put in $M was easy: We put everything we could think of and haven't regretted it. In fact, some non-NIS files are maintained there, including the configuration file that drives our DNS zone generation system. When the DNS configuration files changed, the h2n program from [Albitz1996] is run automatically and the daemon is signalled to load the new configuration. We put other files into $M, such as our host inventory, list of mail servers, and files related to maintaining our Network Appliance file servers.

The makefile encapsulates all the update procedures in one place. This makes the question "where do I add a new feature" moot. The entire system becomes easier to debug.

Rather than maintain duplicate sets of procedures during the transition, we used the new NIS master to drive the old legacy systems. That is, we used the Makefile to encapsulate the confusing procedures of the legacy NIS masters. The legacy systems had scripts to process each kind of updated file and often the script that had to be run had a different name and location on each legacy system. The Makefile "did the right thing." For example, if a change was made to $M/auto_home, the Makefile would copy it to the correct place on the old NIS masters (and even do some translation for one of them) then run the appropriate update scripts on those masters. Eventually this new master was driving all the tables of the old masters. As the old masters lost their clients, we removed the "push to legacy" portions of the Makefile. (See Listing 1)

NIS slaves were configured to serve the NIS databases of the old and new domains at the same time. Many people do not realize that an NIS slave can serve the data of multiple NIS domains at once. [Stern1991] explains why this works and how to configure a slave to do so. Once this was complete, individual clients could be converted to the new NIS domain "at will" since both domains were available on every subnet. This enabled us to convert a small number of machines for testing. It was critical that the new NIS domain served the proper information before "real" users were exposed to them.

Sidenote: When writing the scripts that automated our processes we adopted a new guideline that reversed a previous tradition: Only write the code for the features you will immediately use. Our older scripts (the ones obviously not written by us) were convoluted and difficult to use because most of the code dealt with features the author thought we might use but didn't. This new guideline reduced our maintenance and simplified everything.

Building The Perfect Pair(s)

We learned the hard way that is is better to wait and deploy all changes at once than to deploy each change globally when we thought it was ready. When the new DNS servers were ready we spent two days making sure that /etc/resolv.conf on all machines pointed to the new servers. However, then we realized that one of the servers would have its IP address changed soon. Since /etc/resolv.conf contains IP addresses, not host names, we had to repeat this global change. We couldn't risk this in the future.

We decided instead to make one perfect SunOS client and one perfect SunOS server and move a couple "friendly users" to these machines for testing. Once this was rolling we created one perfect Solaris client and Solaris server.

Once the needed changes were documented, they could be embodied into a script and our 30 or so

changes could be done at once, and we'd know they were all correct. Every time we had to visit a machine to make a change it is a bother to our users. The worst thing we could do would be to visit each machine 30 times to make 30 changes. With this script, we could make all changes in one (virtual or physical) visit and a single reboot.

We brought our users into this process by placing some of these "perfect" machines in public spaces and asked numerous users to log into them to make sure their startup files (.profile, .cshrc, etc.) operated properly. Also, users that were paranoid that something (homegrown tools, etc.) would break in the new configuration were walked to a perfect machine for "real time" debugging and feedback. We did find bugs thanks to these users.

As the configurations stabilized, we cloned everything for AT&T Labs. We made the "one perfect client" for AT&T Labs, built a server, and modified the script to do the right things if it was run on an AT&T machine.

We also set up a clone of our NIS master for AT&T and gave them our new procedures and automation. In fact, until certain files were split, our NIS master pushed to the AT&T NIS master the same way it pushed other data to the legacy NIS masters.

Our "reconfigure" script took two arguments: the new IP addr and hostname; from there it could determine everything else (name of NIS domain, default route, etc. Since the IP address indicated which company the machine was targeted for, even special things needed for particular companies were automatically done.

The "reconfigure" script could handle just about any situation but had to be manually brought to the machine. We did this because becoming root on a machine is different in each legacy area. Some machines accepted rcp/rsh as root from some "master" server. Other machines did not permit us in as root in any of our usual ways and not all machines could be NFS clients. Therefore to execute it you FTP'd a tar file from a central machine, untared the file in /tmp and ran one script (which asked for an IP address and new host name and absolutely nothing else). The tar file contained the right files for AT&T and Lucent, and the script could make all of its decisions from the IP address it was given, including what company's configuration was needed.

We tested the script on machine after machine until it was bug free for every combination of OS – SunOS or Solaris – and company – AT&T or Lucent. The script was hacked to automate some of the changes for less popular machines, like our microvaxen that certain users just refused to let go of.

Dividing The Fileservers

Some file servers had data of users from "the wrong company" that had to be moved. Many file servers were old and the data was moved to new file servers purchased as a result of the tri-vestiture. It was easier to purchase new fileservers than to split old ones. Unlike [Remy1994] we had money to burn, but not people power.

We used netgroups to help our split. Netgroups is a difficult file to edit, and our old system generated one huge netgroup from our /etc/hosts file. Our new system was driven by a meta file ($M/netgroups.config) which "mk.netgroup" (See Listing 2) converted to a proper NIS netgroup file. (Our master "`ypmake`" Makefile did the conversion automatically, of course.)

The format looks like this:

```
allnfs: +att +bl +unknown
machineA: att
machineB: bl
machineC: unknown
machineD: att bl
```

This would generate netgroups for the att machines, the bl machines, and the unknown machines. It would also generate a netgroup called "allnfs" which contained the att, bl, and unknown netgroups. Initially all machines were listed in the unknown group and the fileservers were exporting to "allnfs." As we learned which machines were going to which company, we changed their netgroups. As we eliminated, for example, AT&T files from Lucent file servers, we changed the exports file on the server to export to "bl." The NIS master's Makefile also generated a web page from this file that detailed which machines were in which netgroup so that users could verify our data. This was important because our information about which machine went to which company was spotty and involving users in the process helped dramatically.

We would know this phase was done when all the files were moved to the right servers, all partitions were exported to either "bl" or "att" but not "allnfs," all partitions on a file servers were exported to the same company, and all the machines were classified with a single netgroup. These tasks could be divided over many SAs and done in parallel.

Merge In

Our plan was to run our main 10 subnets and the two new subnets on an array of ethernet switches. Machines configured for any of these 12 IP subnets could be plugged into this network. We would be able to renumber a machine without having to wait for our telecom department to change the machine's jack to be on their new network.

Multiple IP Subnets On One Wire

It is possible to run more than one IP subnet on an ethernet. This works for the same reason that the same ethernet can have machines that talk TCP/IP, Ethertalk (Appletalk over ethernet), and DECNET connected at the same time. Each computer is smart enough to ignore the packets for the other machines. Since this concept is not commonly known, we will explain how it works.

In Figure 1 we see two workstations connected to an ethernet hub as one would expect. They are on subnet 1.1.1.* (netmask of 255.255.255.0). On an ethernet, all hosts see each other's packets but are smart enough to ignore packets not destined for themselves.

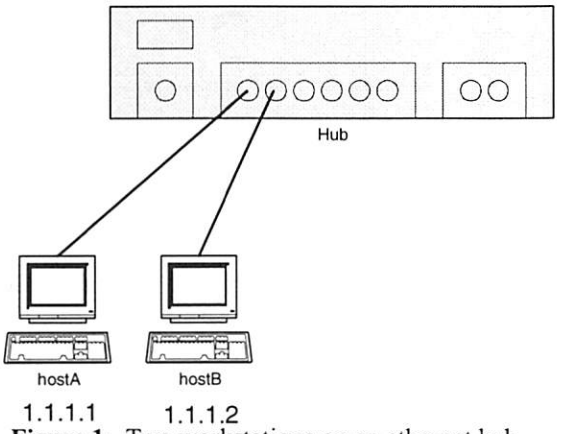

Figure 1: Two workstations on an ethernet hub.

If two machines configured for subnet 2.2.2.* (same netmask) were connected to this hub (see Figure 2), they communicate with each other as one would expect but because the way IP is designed, X and Y would ignore the packets sent between A and B and vice versa. However, host A, B, X and Y share the bandwidth available. That is, while there are two IP subnets, the network is still a shared 10M of bandwidth. That is, while there are two IP subnets on this ethernet, the total bandwidth does not double.

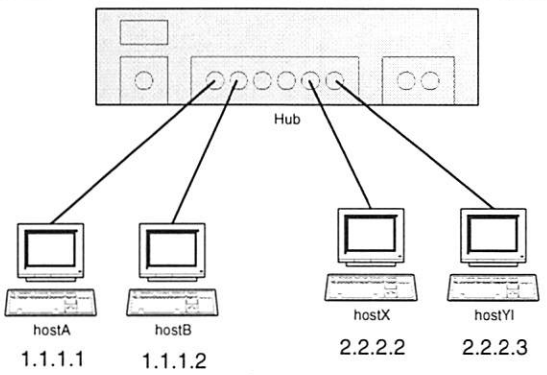

Figure 2: Two subnets on the same hub

Suppose host A wanted to communicate with host X. Normally these two machines would not be on the same hub and host A would send its packet to a router. This router would then deposit the packet on host X's ethernet. In this case, however, host A, X, and the router are all on one ethernet (see Figure 3). There is no magic. Packets still must be sent through a router. In this case, host A will send the packet to the router, who will forward the packet out the same interface it came from (with proper packet header changes, etc.) so that X receives this packet.

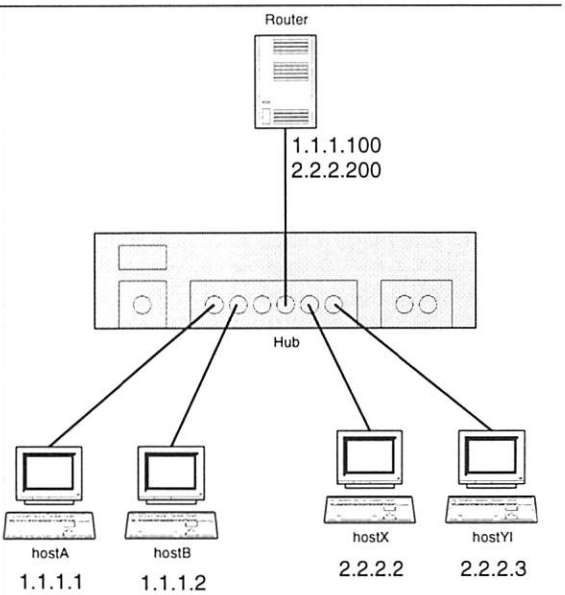

Figure 3: Routing among subnets.

For this to work, the router must be able to assign two IP addresses to the same interface. This is called a "secondary IP address." In our example, host A sent the packet to its default route, 1.1.1.100, and host X received the packet from the router at IP address 2.2.2.200.

The bandwidth used on this ethernet is twice as much as a normal packet since it crossed the ethernet once to get to the router and a second time to get to host X.

Renumbering NFS Servers Made Easy

Changing the IP address of a fileserver means rebooting all of its NFS clients since NFS clients never check to see if a server's IP address has changed. This is because an NFS mount is to an IP address, not a hostname. If we renumbered a NFS server, the clients would hang until they were rebooted, waiting for the NFS server to reappear at the old address.

Unix workstations and servers can be configured to have secondary IP addresses the same as routers. At the time of our split, most of our file servers ran SunOS 4.1.x which does not support secondary IP addresses (Solaris 2.x and IRIX 6.x do[3]). We cleared

[3]See the *ifconfig*(1M) man page

this roadblock by using `vif`[4], a device driver that lets a SunOS machine appear on two IP addresses at once.

We updated our configurations so that all new NFS mounts would be addressed to the server's new IP address. As clients rebooted (for whatever reason), we would eventually have all machines talking to the new IP address. However, now most machines were talking to their main file server through their router. A small price to pay for the ability to renumber clients in a lazy, as needed, fashion. Once all clients had been rebooted, the NFS server is reconfigured to only use the new IP address.

Split Out

The physical split was designed to be neat and orderly after a big merge. To split the networks, we planned on merging the pre-existing production networks into one major network in each building. Since all network jacks could simultaneously support machines of both old and new IP addresses, we could renumber machines at will. Once they were renumbered, we would move machines to the ethernet hub of the appropriate company.

Once the hubs were segregated, we would move the hubs onto the ethernet switches of the appropriate company. Then we could rearrange the connections between the ethernet switches so that they were only connected at once place. Then we would plug the router into the ethernet switches at two places, one for each company.

As with each step of this project, we found there was a fast way to do something that would require downtime and a slower, more creative, method that would reduce downtime to a minimum and let us test as we went along. Our original plan was to merge a network's machines onto the ethernet switches by announcing some downtime which our telecom department would use to reconnect each hub in each closet to the newly installed ethernet switches. We would have to repeat this ten times. (one for each subnet!) We found a better way.

The Lay(out) Of The LAN(d)

We use the Lucent SYSTIMAX Structured Cabling System to wire our building. This is an example of the 10base-T "star" topology that most medium to large buildings use. Each office is wired to a hub in the nearest closet via copper. The hubs in the closets are then connected via fiber uplinks to the basement. If a network spans multiple closets, they all "meet" in the basement at a fiber ethernet hub which connects all the uplinks.

Due to our users' office locations, our networks span about ten of our telecom department's wiring closets. For example, our "quarto-net" looks like Figure 4.

[4] `vif`, originally by John Ioannidis, 1991. Modified by many. See http://fy.chalmers.se/~appro/VIF.html

Our "lexicon-net" connected a different, but overlapping set of closets. (Imagine 10 major networks and 20-odd small networks overlapping in those closets... and our users represent only 10% of the people in this building.)

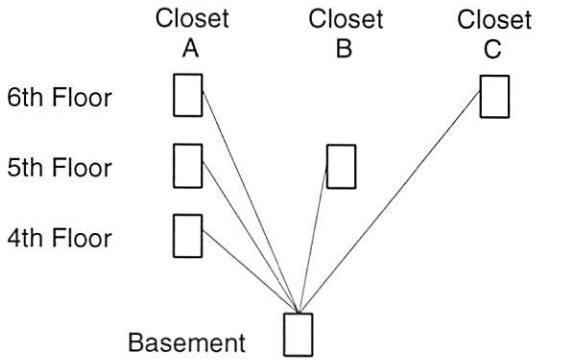

Figure 4: The "quarto-net."

A typical closet looks like Figure 5. This diagram represents a closet with many lexicon-net users and a smaller number of quarto-net users right before we migrate users to the ethernet switch (it is unused, except for its connection to the other switches). The lower lexicon-net hubs connect to the top hub, and the top hub connects to the basement. The three lexicon-net hubs are connected in a star rather than a daisy-chain because the diameter of an ethernet is limited by the number of repeaters (hubs count as repeaters). The quarto-net hubs are in a similar arrangement but the star configuration is less obvious since there are only two.

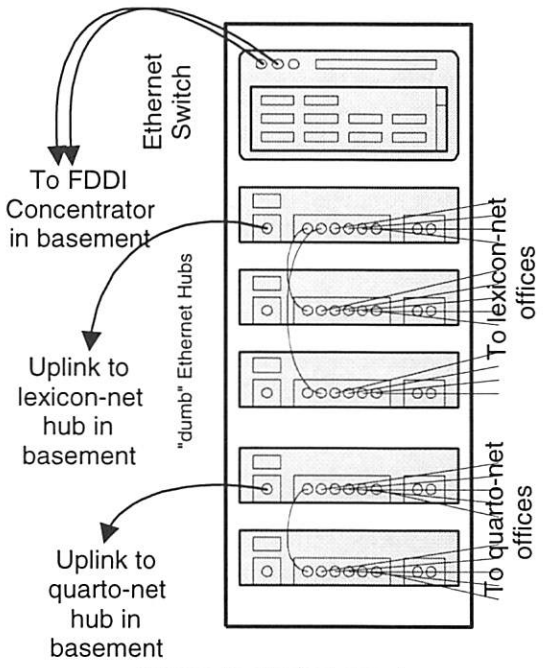

Figure 5: Typical closet.

Migrating To The Ethernet Switch

Here is how we moved users to the ethernet switch with almost no downtime. In the beginning, the switched network (known as "half-net," because we would eventually split it in half) and lexicon-net are independent networks. They are connected by a router that is in a different part of the building. The router connection for half-net is a dedicated port on an ethernet switch. The router connection for lexicon-net and others is like any other office connection on lexicon-net.

First we connect the entire lexicon-net to half-net at one point (Figure 6). We do that by connecting the top hub in one closet to a port on our switch. This is now two IP subnets on one (switched) ethernet. The router connects at two physically different places (one for half-net, one for lexicon-net) and all data flows as before. Downtime for this maneuver: none.

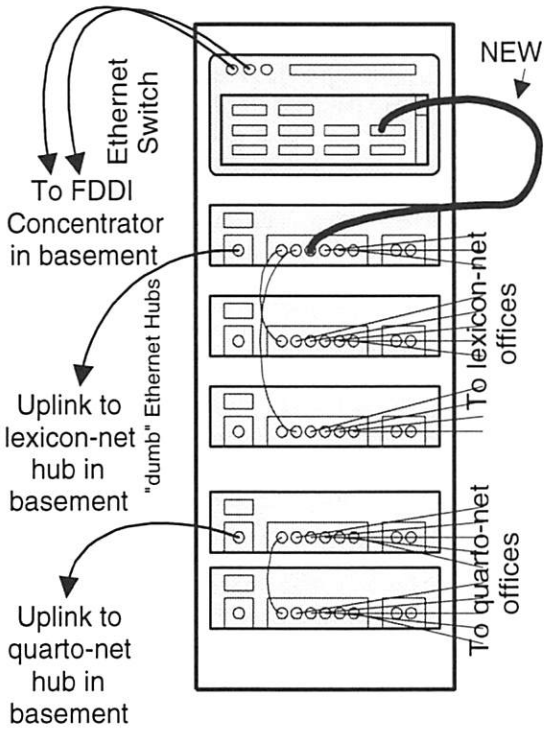

Figure 6: Connection of lexicon-net to half-net.

Now we decommission the router's lexicon-net interface and assign its old IP address to be the secondary IP address of the router's half-net interface. Now both half-net and lexicon-net are connected over the same router interface. During this process the machines on lexicon-net will see their router become inactive, then re-appear with a new MAC address. Most TCP/IP implementations cache a MAC address in the "ARP Cache" for 5 minutes and will hang until this value times out and a new ARP is issued to learn about the new router. Downtime can be reduced if the router and clients implement "Gratuitous ARP." This is where a machine bringing up a new interface broadcasts an ARP packet with the question and answer sections filled in. Most clients will use this information to replace whatever is in their ARP Cache. While clients should implement this, they rarely do. To minimize the distruption, we usually did this at night. It also helped that we manually cleared the ARP caches on the servers and routers right after making the change. Downtime for this maneuver: 5+ minutes

Now that the router has been moved, we are free to move all other hubs onto the ethernet switch at any pace using the following algorithm:

1. Disconnect the hub from the rest of lexicon-net.
2. Connect it to the ethernet switch.

It is important to disconnect the hub before you connect it to the switch, or you may create a loop. Since some hubs (the top hub in our figure) are connected to lexicon-net three times (it is used to connect all the other lexicon-net hubs in that closet) it is best to process that hub last. Downtime for this maneuver: 10 seconds per hub, plus one second for the ethernet switches to "find" the change.

Once the first step of this began, users machines could be renumbered. If this migration took a long time to complete it could be done in parallel with the renumbering of the clients. Parallelism is good. Once the router was reconfigured, our telecom department could re-connect the hubs at will. A lot was gained by not having to move lock-step with our telecom department; they were extremely busy with the other networks in the building.

Now that we could renumber any machine at will. We renumbered each to either the new AT&T subnet or the Bell Labs subnet. We used the techniques described later in this paper.

Splitting Hairs

The renumbering took two months and during that time we we collected information about which jacks had machines targeted to AT&T and which had machines targeted to Lucent. We correlated employees targeted to AT&T to their office number, and office numbers with jacks. Who was going to which company was a moving target, but by enlisting the secretaries the process went exceptionally smoothly.

Also during this time our telecom department installed additional ethernet switches in the closets that would have both AT&T and Lucent networks. These AT&T ethernet switches were connected to their own FDDI concentrator, but the two (AT&T and Lucent) FDDI concentrators were connected to make one large ring.

We delivered our list of which jacks were targeted to which company to our telecom department. They set to work grouping users so that hubs connected exclusively AT&T or Lucent endpoints. When they were done a typical closet looked like Figure 7.

The Grand Finale

When the AT&T router had arrived, we would disable the AT&T net secondary IP address on our router and configure AT&T's new router to use that IP address. The routers were connected by a short wire that we could break when ready. Eventually the AT&T router had its own connection to the Lucent/AT&T backbone and the wire wasn't needed. Around the same time, we separated the AT&T FDDI ring from the Lucent FDDI ring. That was the moment of truth when we would find out which machines were not properly renumbered, and which jacks were not connected to the correct hubs. Much to our surprise, we had exactly four machines lose connectivity when we made that last cut. We considered that a huge success!

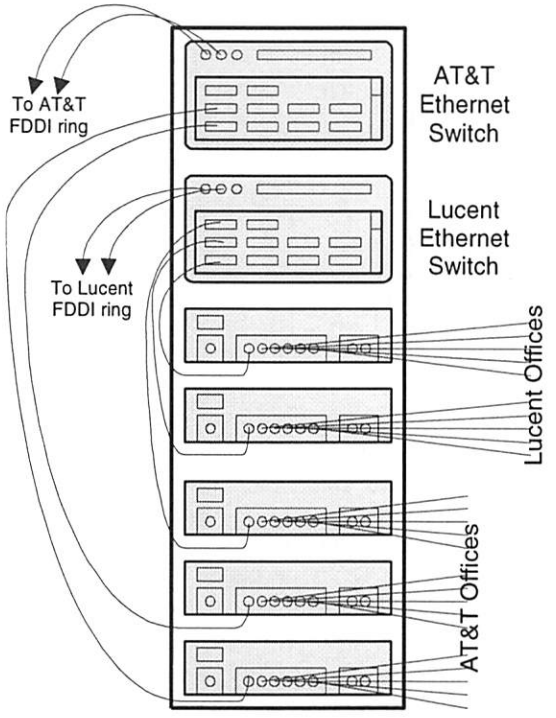

Figure 7: Completed hub schematic.

Then It All Fell Apart

We had some glitches. Some wires were connected incorrectly; sometimes tugging on one wire made another come loose. The subnets of a Class B network must be contiguous, and since the AT&T router for Crawford Hill arrived late, we ended up having to complete the Crawford Hill building first before we could do parts of the Holmdel split. However, none of these problems compared to what happened next.

After the big merge, things really fell apart. Some Sun servers weren't reliable with the `vif` driver, so we used spare ethernet cards to give the machines two physical ports. Sun workstations assign the same MAC (ethernet) address to both interfaces.[5] This confused our ethernet switches and once actually caused a Cabletron ESX-MIM to hang until it was power-cycled. Since it was in a closet owned by our telecom department, we couldn't reboot it until they arrived in the morning.

The big problem was that we underestimated the need of bandwidth to our router. When using secondary IP addresses, packets destined for a different IP subnet (but the same physical ethernet) go up to the router then back down the same router port along the same physical ethernet. While we realized that this would mean an extra load on the router, the load was too heavy and the network fell apart. Machines were nearly unusable if they had to access a fileserver via the router. They all did. We felt that dedicating multiple switched ports to the router would solve the problem but it didn't. (Neither the router nor ethernet switch vendor could explain why, but they both kept explaining that it should work.)

We don't know why we didn't expect this and upgrade the bandwidth to the router, but the fact is that we didn't, and we paid heavily for this. Now FastEthernet is commonplace, but it wasn't then. In hindsight we realize that every packet from every non-renumbered client would be going through the router. That would be 90% of all packets at the beginning of the conversion! Obviously, that is more than a 10M ethernet can handle.

The Trouble Began

Users were extremely upset and got management involved. They demanded we fix the problem before we moved on. However, we felt the best solution was to move on because completing the renumbering would fix the problem and serve the need to move the project forward at the same time. We went back and forth on this and management started hinting that they wanted to see proof that the final configuration would work at all. This would be impossible because we had no way to prove that except with sketchy Sniffer plots and hand waving.

We did not want to take any steps backwards because we felt it was difficult enough to make progress and if we permitted one slide backwards to fix a problem, we would end up sliding back in other ways and eventually the split would not be completed at all; then corporate management would be really upset.

We call this "The Broken Network Conundrum." This is where you have to decide between fixing things and explaining to users why they don't work, because you don't have time for both. One of us

[5]The standards are unclear on how multihomed hosts should behave in this situation. Sun assigns the same MAC address to all ethernet ports of a host. Other vendors assign different MAC addresses per interface.

often muttered, "I'm too busy mopping the floor to turn the faucet off." This was about the time that many of us were ready to quit.

The result is that we spent about three weeks with a network that was unusable to many of our users. During that time we had meetings with vendors, generated reports with sniffers, chased non-problems, etc. Meanwhile, we renumbered the users that complained heavily so that they were on the same IP subnet as the machines they accessed, therefore eliminating the problem for them. Eventually the complainers managed to all get their machines renumbered, which was our original proposal for fixing things.

Finally we could move on.

Storming The Hallways

We were permitted to continue once we reached a point where the intolerable complainers had been satisfied and the others began to understand that moving forward would fix the problem. At this point they had needlessly spent three weeks with a nearly unusable network. Conclusion: once you renumber the servers, renumber the clients immediately. We were so exhausted after renumbering the servers that we didn't begin the clients right away. Considering that certain groups of clients communicated mainly with certain servers, we could have done the renumbering one cluster at a time (the server followed by its clients).

We announced a calendar (Listing 3) of when each hallway would be converted. Each week consisted of converting a number of hallways on Monday and Wednesday, giving us a day in between to fix problems that arose. We made no changes on Friday in hope that we might sleep easy on the weekends. We warned users that their hallway's date would only be changed if they could convince another hallway to swap with them. Most hallways were almost entirely the same department and gladly accepted the calendar as an excuse to plan alternative activities. One department held a picnic.

On the days set aside for changes, we used what we called "The Rioting Mob Technique." At 9 A.M. we would stand at one end of the hallway. We'd psych ourselves up, and move down the hallways in pairs. At each office we kicked the users out of the office and went machine to machine making the needed changes. Two pairs were PC admins, two pairs were Unix admins. (Each pair either did the left or right side of the hallways). The Unix script was quite robust but sometimes broke, or the tar file was too large for /tmp, or becoming "root" on the machine was difficult. Rather than trying to fix it themselves, the SA would call the senior SA that wrote the script to fix the problem and move on to the next machine. Meanwhile a final pair of SAs stayed at our "command central" where people could phone in requests for IP addresses, provide updates to our host inventory, the host table, etc.

We spent the next day cleaning up anything that had broken. On this "breather" day we also met to refine the process. After a brainstorming session determined what went well and what needed improvement, we determined it was better to make one pass through the hallway calling in requests for IP addresses, giving users a chance to log out and identifying non-standard machines for the senior SAs to focus on. The second pass through the hallway everyone had the IP addresses they needed and things went more smoothly. Soon we could do two hallways in the morning and do our cleanup in the afternoon.

The brainstorming session between the first and second conversion day was critical as everyone had excellent suggestions about how to improve our process. On subsequent breather days we still met but now that our process was refined, our meeting was used to pre-plan for the next day. Many times a conversion day went smoothly enough that we were done by lunch, had the problems resolved by the afternoon, and spent our breather day on other projects.

Other Changes

Meanwhile many other systems needed to be cloned, moved or functions disbanded. This included email, news and many DNS-related issues. These issues could fill another paper. We will not document them because they are very specific to our site and most of what we did was non-inventive.

Communication Is Key

We held weekly "user feedback sessions" to answer questions and give status and "heads up" information. This made users feel included in the process, which increased their cooperation. They also provided excellent feedback about what they felt was important.

Reward Those Who Helped

Eventually, the IP portion of the split was complete. We still had to split the passwd files, duplicate some license servers, and clean up a million small issues before the corporate routers would no longer pass packets between the two companies. But the big physical network split was finally over. We felt it was important to reward those that helped with the project. Our management surprised the technicians from the telecom department by awarding them bonuses. Even though we were only 1/10th of their users in this building and the other 90% were less technically sophisticated and required them to do a lot more of the ground work, we believe we were the only ones to reward them so. Later both AT&T Labs and Lucent Bell Labs both rewarded us similarly.

Related Topics

The PC Movement

DHCP saved us. When we stormed the hallways the PC SAs usually simply clicked "Use DHCP" and rebooted. DHCP is a protocol extension to BOOTP which permits machines to get just about every network parameter they might need from the network. This lets us maintain a central database of parameters which can be updated globally. PCs receive their updated parameters the next time they reboot. We wished Unix could use DHCP so well.

NCD X Terminals

Our NCDs had four "classes" of configurations (one of which being "every man for himself"). We developed one solid and maintainable configuration that worked everywhere and changed all NCDs to use it. We now store no configuration on the terminal and use TFTP for all the data. Each client's configuration file is simply a symbolic link to the master configuration file. We can make a global change by modifying the master file and waiting for all the NCDs to be rebooted. A simple script was written to create the right symbolic links so that the error-prone task of converting IP addresses to hex, and typing the extremely long path names was eliminated.

What We Would Do Differently

In hindsight, there are some things we could have done differently. We could have renumbered just the AT&T machines, not all of them. We (Lucent) were the majority. However, this option would not have let us use this opportunity to renumber to our new Class B IP network, do our much-needed Unix re-configuration, and upgrade our network performance, etc. However, it would have reduced our work to almost nothing more than changing our DNS and NIS domains. However, it was not an obvious option to us because AT&T was short staffed and we felt obligated to back-fill for them. We would have had to do much of the work anyway.

We also could have simply declared ownership of the network and gave AT&T a cut-off date for when they had to have all their machines removed. Bell Labs Murray Hill used this technique . . . the lucky dogs! Again, AT&T's staff shortage would have made this a problem.

"At least it won't happen again"

We felt oddly disappointed that we had learned so much and developed so many techniques but none of them would be useful in the future. However, as luck would have it, since the split we have found that soon we will need to renumber three more user communities, totaling approximately the same number of machines involved in the split. Imagine our joy and surprise.

What We Learned

- Massive renumbering projects are increasing in frequency due to cleansing of organic networks and corporate structural changes.
- Massive renumbering projects are an excellent opportunity to clean up a messy network.
- It is possible to renumber a massive number of machines without a single day (or time period) where all machines are down.
- After the first day of mass conversion, meet to review areas of improvement and update the process.
- Secondary IP addresses are a useful transition aid, but don't over-do it.
- Clients have dependencies on servers, so be creative about renumbering servers, and do them early (or add secondary IP addresses).
- Develop "the perfect machine" before you make those changes everywhere else. Make all changes at once to a machine before you move on, don't make one change to all machines before you move to the next change. This prevents the situation where you convert all machines at once to discover that every single machine now has the same flaw.
- Communicate with your users any way and every way you can.
- The physical split of the network can be easier if you have a solid, structured, wire plant; you know the structure is good when it documents itself.
- Avoid "the conundrum"; trust your gut.
- Automated processes should be as simple as needed, but no simpler. Centralize things that should be centralized, but no 'centraler'.
- Yell a loud chant before you storm the hallways. It psyches you up and makes your users more willing to get out of the way.

Conclusion

On the network side, we used secondary IP addresses to ease the renumbering of the networks. On the management side, we unified our management files so updates are now easy and error-proof (the $M directory and "`ypmake`" script). On the software (operating system) side, we came up with a good automated method of implementing this split, with a larger purpose in mind: our "reconfig" script is still used to change the configuration of SunOS and other systems that don't support Jumpstart. The "setupncd" script lives on. On the human side, we used communication (weekly meetings/forums, web pages, email broadcasts), and terror ("the rioting mob technique").

"Normal" work for us nearly ended for nine months as we split the network. We did not take one mess (which every large network grows into) and make it two separate messes. Instead we actually took a tough task (splitting the network) and in the process

of accomplishing it, made things better. These were not two simultaneous but unrelated events. Instead, we came up with a means to do them simultaneously in such a way that they complement and ease the difficulty of each other. The new network is significantly less labor-intensive to manage because we were able to put in place unified policies and procedures.

Acknowledgements

Huge projects like this do not get completed without the help and dedication of the entire team or in this case two teams: all the system administrators involved both from AT&T and Lucent. Our telecom department performed great feats of heroism, especially Dave Wilson. Our significant others and families deserve credit for enduring our long hours and frazzled nerves. We'd also like to thank the users that suffered through it all. Special thanks to Sherry McBride and Mike Richichi for their advice on how to structure this paper.

Availability

Some of the tools created for this project are available on http://www.bell-labs.com/user/tal

References

[Albitz1996] *DNS and BIND*, By Paul Albitz and Cricket Liu, 2nd Edition December 1996, O'Reilly and Associates.

[Hess1992] David Hess, David Safford, Udo Pooch, "A Unix Network Protocol Security Study: Network Information Service", *ACM Computer Communications Review* 22 (5), 1992. ftp://net.tamu.edu/pub/security/TAMU.

[Lear1996] "Renumbering: Threat or Menace?" Eliot Lear, Jennifer Katinsky, Jeff Coffin, & Diane Tharp, Silicon Graphics, Inc., published in the *USENIX Association's Proceedings of the Tenth Systems Administration Conference (LISA X)*, pp. 91-96.

[Remy1994] "Tenwen: The Re-engineering of a Computing Environment", Remy Evard, *Proceedings of the Eighth USENIX Systems Administration Conference Proceedings*, pages 37-46, 1994.

[RFC1900] *RFC1900, Renumbering Needs Work*, B. Carpenter & Y. Rekhter. February 1996.

[RFC1916] *RFC1916: Enterprise Renumbering: Experience and Information Solicitation*, H. Berkowitz, P. Ferguson, W. Leland, & P. Nesser. February 1996.

[RFC2071] *RFC207: Network Renumbering Overview: Why would I want it and what is it.* P. Ferguson, H. Berkowitz. January 1997.

[Stern1991] *Managing NFS and NIS*, Hal Stern, June 1991, O'Reilly and Associates.

[Wietse1996] "The Replacement Portmapper", Wietse Venema, <ftp://ftp.win.tue.nl/pub/security/portmap_*>, 1996.

Author Information

Tom Limoncelli is a MTS at Bell Labs, the R&D unit of Lucent Technologies, where he is chiefly concerned with the architecture and operation of the data network for much of Research. Tom started doing system administration on VAX/VMS systems in 1987 and switched to Unix in 1991. He holds a B. A. in C. S. from Drew University, Madison, New Jersey. His homepage is http://www.bell-labs.com/user/tal; he can be reached at <tal@lucent.com>.

Tom Reingold studied music (voice, performance) at Boston University from 1978 through 1980 and computer science at Hunter College of the City University of New York from 1982 through 1987. He has worked both as a programmer and as a system administrator in a variety of environments, ranging from business support, commercial software development, and research. He currently works as a system administrator for the Software and Systems Research Center at Bell Labs. His homepage is http://www.bell-labs.com/user/tommy; he can be reached at <tommy@bell-labs.com>.

Ravi Narayan is a contractor at Lucent Technologies Bell Labs, working on systems and networks administration. He is finishing up his Master's in Computer Science at Worcester Polytechnic Institute, and holds a Bachelor's in Mathematics. His interests include Distributed Systems and Networking, automation of administration, hacking in Perl and the World Wide Web (authoring and administration, since 1993, and mentioned in technical magazines such as PC Computing: 1001 Best Web Sites, Dec 94). His homepage is http://www.bell-labs.com/user/rn; he can be reached at <rn@bell-labs.com>.

Ralph Loura is a Technical Manager at Bell Labs Research, the Research arm of Lucent Technologies, where he is responsible for managing the team providing computing and networking services to about half of Bell Labs Research. Ralph has been working with UNIX Systems since the mid eighties in jobs that have included O. S. programing, device driver development, distributed applications programming, systems administration, systems engineering, computing environment architecture and strategy, vendor management and financial planning. He holds a B. S. in C. S. and Mathematics from Saint Joseph's College, Rensselaer, IN and a M. S. in C. S. from Northwestern University, Evanston, IL. His homepage is http://www.bell-labs.com/user/ralph; he can be reached at <ralph@lucent.com>.

Listing 1: The NIS Master's Makefile

```
# The NIS Makefile
# Master copy: milk:/var/yp/Makefile
# Source Control: RCS
# Distribution: none (it stays here)

[much deleted]

B=-b
#B=
#VERBOSE=
VERBOSE=-v
#DIR =/etc
DIR =/var/yp/master
DOM = `domainname`
NOPUSH = ""
#NOPUSH = 1
#ALIASES = /etc/aliases
ALIASES = $(DIR)/aliases
YPDIR=/usr/etc/yp
YPDBDIR=/var/yp
YPPUSH=$(YPDIR)/yppush $(VERBOSE)
MAKEDBM=$(YPDIR)/makedbm
REVNETGROUP=$(YPDIR)/revnetgroup
STDETHERS=$(YPDIR)/stdethers
STDHOSTS=$(YPDIR)/stdhosts
MKNETID=$(YPDIR)/mknetid
MKALIAS=$(YPDIR)/mkalias

# Local defines
DNSCONVERT=/var/named/bin/h2n
MKNETGROUP=/usr/local/adm/bin/mk.netgroup
NETGROUPCONF=$(DIR)/netgroup.config

CHKPIPE=  || (  echo "NIS make terminated:" $@ 1>&2; kill -TERM 0 )

k:
    @if [ ! $(NOPUSH) ]; then $(MAKE)  $(MFLAGS) -k all; \
    else $(MAKE) $(MFLAGS) -k all NOPUSH=$(NOPUSH);fi

all: passwd group ethers hosts web.done networks rpc services protocols \
    netgroup.time bootparams aliases netmasks \
    auto.master auto_master auto.home auto_home timezone \
    ypservers.time hosts.mail.time dns.time toaster.time netid \
    legacy.time attlabs.time

[ the following are not deleted as they are unchanged from
the sample that comes with SunOS 4.1.4: passwd.time, group.time,
hosts.time, ethers.time, networks.time, services.time,
rpc.time, protocols.time, netgroup.time, bootparams.time,
aliases.time, netmasks.time, netid.time, timezone.time ]

auto.master.time:  $(DIR)/auto.master
    -@if [ -f $(DIR)/auto.master ]; then \
        sed -e "/^#/d" -e s/#.*$$// $(DIR)/auto.master \
        | $(MAKEDBM) - $(YPDBDIR)/$(DOM)/auto.master; \
        touch auto.master.time; \
        echo "updated auto.master"; \
        if [ ! $(NOPUSH) ]; then \
            $(YPPUSH) auto.master; \
            echo "pushed auto.master"; \
        else \
        : ; \
```

```
        fi \
    else \
        echo "couldn't find $(DIR)/auto.master"; \
    fi

[ the following maps are not included because they are extremely
similar to auto.master.time: auto_master.time, auto.home.time,
auto_home.time ]

# The NIS servers are listed in $M/ypservers
# (Now we can edit $M/ypservers rather than using "ypinit -m"
# to edit our list of slaves)
ypservers.time:  $(DIR)/ypservers
    -@if [ -f $(DIR)/ypservers ]; then \
        sed -e "/^#/d" -e s/#.*$$// $(DIR)/ypservers \
        | egrep -v "^\s*$$" \
        | (awk '{ print $$1 " " $$1 }') \
        | $(MAKEDBM) - $(YPDBDIR)/$(DOM)/ypservers; \
        touch ypservers.time; \
        echo "updated ypservers"; \
        if [ ! $(NOPUSH) ]; then \
            $(YPPUSH) ypservers; \
            echo "pushed ypservers"; \
        else \
        : ; \
        fi \
    else \
        echo "couldn't find $(DIR)/ypservers"; \
    fi

# This generates DNS data from the NIS hosts file.

dns: dns.time
dns.time: $(DIR)/hosts $(DIR)/dns.opts
    (cd /var/named && make)
    touch dns.time

# This generates netgroup data from the $M/netgroup.config file

$(DIR)/netgroup: $(DIR)/hosts $(DIR)/netgroup.config $(MKNETGROUP)
    $(MKNETGROUP) -d sun1134      ${NETGROUPCONF} >$(DIR)/netgroup.sun1134
    $(MKNETGROUP) -d sun1135      ${NETGROUPCONF} >$(DIR)/netgroup.sun1135
    $(MKNETGROUP) -d hoh          ${NETGROUPCONF} >$(DIR)/netgroup.hoh
    $(MKNETGROUP) -d ''           ${NETGROUPCONF} >$(DIR)/netgroup.toaster
    $(MKNETGROUP) -d info.att.com ${NETGROUPCONF} >$(DIR)/netgroup

toaster.time: toaster.host.time toaster.netgroup.time
    touch toaster.time

# Push hosts to the NAS FAServers
toaster.host.time: $(DIR)/hosts
    mntdir=/tmp/nac_etc_mnt_$$$$;\
    if [ ! -d $$mntdir ]; then rm -f $$mntdir; mkdir $$mntdir; fi;\
    for faserver in `cat $(DIR)/toasters` ; do \
        echo Pushing hosts to $$faserver;\
        mount $$faserver:/etc $$mntdir;\
        cp $$mntdir/hosts $$mntdir/hosts.bak;\
        cp $(DIR)/hosts $$mntdir/hosts.new;\
        mv $$mntdir/hosts.new $$mntdir/hosts;\
        umount $$mntdir;\
    done;\
    rmdir $$mntdir
    touch toaster.host.time
```

```
# Push netgroup to the NAS FAServers
toaster.netgroup.time: $(DIR)/netgroup
    mntdir=/tmp/nac_etc_mnt_$$$$;\
    if [ ! -d $$mntdir ]; then rm -f $$mntdir; mkdir $$mntdir; fi;\
    for faserver in `cat $(DIR)/toasters` ; do \
        echo Pushing netgroup to $$faserver;\
        mount $$faserver:/etc $$mntdir;\
        cp $$mntdir/netgroup $$mntdir/netgroup.bak;\
        cp $(DIR)/netgroup.toaster $$mntdir/netgroup.new;\
        mv $$mntdir/netgroup.new $$mntdir/netgroup;\
        umount $$mntdir;\
        rsh -n $$faserver exportfs -av;\
    done;\
    rmdir $$mntdir
    touch toaster.netgroup.time

# Push the hosts.mail file

hosts.mail.time: $(DIR)/hosts.mail
    rcp $(DIR)/hosts.mail octavo:/etc/hosts.mail
    touch hosts.mail.time

passwd: passwd.time
group: group.time
hosts: hosts.time
ethers: ethers.time
networks: networks.time
rpc: rpc.time
services: services.time
protocols: protocols.time
bootparams: bootparams.time
aliases: aliases.time
netid: netid.time
netmasks: netmasks.time
timezone: timezone.time
auto.master: auto.master.time
auto_master: auto_master.time
auto.home: auto.home.time
auto_home: auto_home.time
$(DIR)/netid:
$(DIR)/timezone:
$(DIR)/auto.master:

# Push things to the legacy systems
#    (this is at the end of the file to make it easy to delete...
#      this will be deleted someday, right?)

legacy.time: legacy.hosts.time legacy.aliases.time \
    legacy.auto_master.time legacy.auto.home.time legacy.auto_home.time \
    legacy.netgroup.peerless.time legacy.netgroup.boole.time \
    legacy.netgroup.octavo.time
    rsh -n octavo "cd /etc && /bin/make"
    touch legacy.time

legacy.hosts.time: $(DIR)/hosts
    rcp $(DIR)/hosts octavo:/etc/hosts.common
    touch legacy.hosts.time

legacy.aliases.time: $(ALIASES)
    rcp $(ALIASES) octavo:/etc/aliases.common
    rcp $(ALIASES) boole:/etc/aliases
    rcp $(ALIASES) learnx:/etc/aliases.common
    rcp $(ALIASES) caribbean:/etc/mail/aliases.common
```

```
	rsh caribbean "cd /etc/mail && /usr/ccs/bin/make install"
	touch legacy.aliases.time

legacy.auto_master.time: $(DIR)/auto_master
	rcp $(DIR)/auto_master octavo:/etc/auto_master
	touch legacy.auto_master.time

legacy.auto.home.time: $(DIR)/auto.home
	rcp $(DIR)/auto.home octavo:/etc/auto.home
	touch legacy.auto.home.time

legacy.auto_home.time: $(DIR)/auto_home
	rcp $(DIR)/auto_home octavo:/etc/auto_home
	touch legacy.auto_home.time

legacy.netgroup.peerless.time: $(DIR)/netgroup.hoh
	rcp $(DIR)/netgroup.hoh peerless:/etc/netgroup
	touch legacy.netgroup.peerless.time

legacy.netgroup.boole.time: $(DIR)/netgroup.sun1134
	rcp $(DIR)/netgroup.sun1134 boole:/etc/netgroup
	touch legacy.netgroup.boole.time

legacy.netgroup.octavo.time: $(DIR)/netgroup.sun1135
	rcp $(DIR)/netgroup.sun1135 octavo:/etc/netgroup
	touch legacy.netgroup.octavo.time

# Push things to the attlabs systems then do their "ypmake"
#    (this is at the end of the file to make it easy to delete...
#      this will be deleted someday, right?)

attlabs.time: attlabs.hosts.time attlabs.ethers.time \
	attlabs.auto_home.time attlabs.netgroup.config.time
	rsh -n shy "cd /var/yp && /usr/local/bin/flock \
	    /var/tmp/ypmake.lock /usr/bin/make"
	touch attlabs.time

attlabs.hosts.time: $(DIR)/hosts
	rcp $(DIR)/hosts shy:/var/yp/master/hosts
	touch attlabs.hosts.time

attlabs.ethers.time: $(DIR)/ethers
	rcp $(DIR)/ethers shy:/var/yp/master/ethers
	touch attlabs.ethers.time

attlabs.auto_home.time: $(DIR)/auto_home
	rcp $(DIR)/auto_home shy:/var/yp/master/auto_home
	touch attlabs.auto_home.time

attlabs.netgroup.config.time: $(DIR)/netgroup.config
	rcp $(DIR)/netgroup.config shy:/var/yp/master/netgroup.config
	touch attlabs.netgroup.config.time

# This generates the web page that lists which machines
# are targeted to which company

web:web.done
web.done: /home/adm/bin/mk.tco-webpages $(DIR)/hosts $(DIR)/tco.machines
	/home/adm/bin/mk.tco-webpages | \
	    rsh xxx "cat >/home/tal/public_html/tco.html"
	touch web.done
```

Listing 2: mk.netgroup

```
#!/opt/perl5/bin/perl

# mk.netgroup -- convert the netgroup.config file to NIS's netgroup format.
#  by tal

# Note: a lot of this comes from THE PERL DATA STRUCTURES COOKBOOK
# http://www.perl.com/perl/pdsc/index.html (esp. "Hashes of Lists" examples)

$MAX_LINE = 252;    # max linelength of a NIS data record

# Process options

$opt_d = '';    # NIS domain.
$opt_m = '';    # special NIS-less rpc.mountd (like on volume)
require "getopts.pl";
do Getopts('d:m');

# read from file data structured like:
# flintstones: fred barney wilma dino
while (<>) {
    # the order of the next three lines is very important
    s/#.*//; # toss comments
    next if /^\s*$/; # skip blank lines
    redo if s/\\$// and $_ .= <>; # handle continuations

    next unless s/^(.*?):\s*//; # parse (and skip badly-formatted lines)
    $host = $1;
    die "Host/group $host is listed twice.  Aborting." if $seen{ $host }++;
    foreach $item ( split ) {
        # is this a group or a host
        if ($item =~ /^\+/) {
            $item =~ s/^\+//g;
            push @{ $netgroup{$host} }, $item;
            # add $item to %GROUPS
            $GROUPS{ $host } = 1;
            $GROUPS{ $item } = 1;
        } else {
            # add the host to that list
            if ($opt_m) {
                push @{ $netgroup{$item} }, "$host";
            } else {
                push @{ $netgroup{$item} }, "($host,,$opt_d)";
            }
            # record that such a group exists
            $GROUPS{ $item } = 1;
        }
    }
}

# write out the netgroup file
foreach $group ( sort keys %netgroup ) {
    &one_group( 0, $group, sort @{ $netgroup{$group} } );
}

exit 0;

# This takes a netgroup name and a list of what should appear
# on its line and prints it.  If the line is too big, it splits
# it into "group_1", "group_2", etc. and generates
# "group: group_1 group_2", etc.
sub one_group {
    local($count, $g, @l) = @_;
    local($line);
```

```
    $origcount = $count;
    local(@m) = ();
    $max = $MAX_LINE - length($g) - 5;
    $line = shift @l;
    foreach $i ( @l ) {
        if ((length($line) + length( $i )) > $max) {
            $count++;
            print $g, "_", $count, "\t", $line, "\n";
            push @m, "${g}_$count";
            $line = $i;
        } else {
            $line .= " " . $i;
        }
    }
    if ($count ne $origcount) {
        $count++;
        print $g, "_", $count, "\t", $line, "\n";
        push @m, "${g}_$count";
        &one_group( $count, $g, @m);    # LOOK!!! RECURSION!!!
    } else {
        print $g, "\t", $line, "\n";
    }
}
```

Listing 3: The Announcement

```
To: (all users)
From: help@big.att.com
Subject: Holmdel/Crawford Hill network split is coming! (READ THIS!!!)

Summary:  We will be stopping by to reboot your machine based on the
schedule below.  No exceptions.  It is required for the Lucent/AT&T
Split.  After your machine has been touched, email from it will either
be "From: user@research.att.com" or "From: user@dnrc.bell-labs.com"
depending on what company the machine is targetted to.  All other
changes should be transparent.

IF YOU WILL NOT BE IN ON YOUR SCHEDULED DAY, PLEASE LOG OUT
THE NIGHT BEFORE.  DO NOT LEAVE YOUR SCREEN LOCKED.

We will be attacking machines on a hallway-by-hallway basis according
to this schedule:

Mon 6/10    5th floor B-E Aisles
Tue 6/11    "fishnet" (5th floor G Aisle)
Wed 6/12    3th floor G Aisle ("neural net")
Mon 6/17    6th floor A-G Aisle
Mon 6/18    "signal-net" users (4th floor)
Wed 6/19    5th floor F-G Aisles ("boole net")
Mon 6/24    All of HOH
6/5-6/15    Spin net users in dept1136 will be spread over these days.

The details:

Before we can split the "big" network (also called the "info.att.com"
network), we must prepare each machine.  This preparation is mandatory
and must be completed by June 26.  It requires that we reboot your
machine.  It will go fastest if we do one hallway at a time.

When we are done you should have as much functionality as you did
before.  The one thing you will notice is that email sent from that
machine will have a From: header that lists
either "From: user@research.att.com" or "From: user@dnrc.bell-labs.com"
```

depending on what company the machine is targetted to. (you will receive email with more details)

If you find something no longer works we need you to report it to "help@big.att.com" as soon as possible or stop us while we are in your hallway converting other machines. You can also call:
HO 908-949-xxxx, 908-949-xxxx, 908-949-xxxx
HOH 908-888-xxxx

We expect some network instability during this process. Be forewarned.

There is a chance that your password may revert back to an older password, if this happens contact us and we can fix this quickly.

NOTE TO LUCENT EMPLOYEES: Lucent users can log into "shelf" (a Solaris 2.5 Sparc 20) or "hyper" (a SunOS 4.1.4 Sparc 1+) to see what a converted machine is like. You shouldn't feel any differences. Please report any problems you find, etc.

NOTE TO AT&T EMPLOYEES: We will soon announce machines that you can log into to see that the new environment works for you.

NOTE TO 1136 "SPIN" EMPLOYEES: You will be contacted individually about your conversion. Access to "shelf" and "hyper" (mentioned above) for you will be ready soon.

VOLUNTEER REQUEST: We would like to convert some machines a couple days early so we'll discover problems specific to your department/area/group. If one courageous Sun user and one dare-devil PC user from each hallway would like to be an "early adopter" it will save the rest of your department a lot of pain. Send email to "help" to volunteer.

ARE WE JUST REBOOTING? CAN I DO THAT MYSELF? No, we're doing a lot more than rebooting your machine. We're installing new sendmail configurations, DNS, automount, and a whole heck of a lot more.

Pinpointing System Performance Issues

Douglas L. Urner – Berkeley Software Design, Inc.

ABSTRACT

The explosive growth of the Internet in recent years has created a heretofore unprecedented demand for system performance. In many cases, system administrators have been left scrambling, trying to understand the performance issues raised by the loads and new technologies they are facing.

This paper suggests methods for analyzing system performance potential, prioritizing the search for bottlenecks and identifying problem areas. While many of the specific tuning suggestions are particular to BSD/OS and other 4.4BSD derivatives, the principles on which they are based should generalize well to any system being used to provide Internet services.

Introduction

This paper discusses methods for tracking performance problems, specific parameters that can be tuned, and data analysis tools available on almost all Unix systems and some other systems, as well.

Starting with a study of the overall architecture, this paper covers configuration, kernel tuning, disk subsystem analysis, networks, memory usage, application performance, and network protocol behavior.

The Big Picture

Methodology

A bit of method goes a long way: it saves time; it gets to the source of the problem quickly; and it helps avoid missing the problem. When trying to improve performance, start where the biggest gains can be had. It makes no sense to speed a system by a factor of 1.01 when a speedup of 5x can be gained. In fact, given today's rules of hardware speeds ever-increasing (at a constant cost) over time, it makes little sense to spend time on 1% speedups. On the other hand, mistuned systems might run at only 10% of their potential speed. Concentrating on the factors that can gain back the 90% is obviously far more productive than concentrating on 1% gains. Of course, figuring out which part of a system gets back the other 90% is the problem.

Overall Architecture

When a system's overall design is wrong, performance can really suffer. The overall design must be right if a system is to achieve its potential. Be sure that your assumptions about the system's use and potential are clear.

Consider the model used by many Unix systems for network demons. The server forks a process to handle each network request it receives. This works reasonably well when the connections are infrequent or relatively long-lived (at least a handful of interactions). But, when connections are short-lived (a single interaction or two), the cost of the fork() can dominate the cost of handling the connection. This is too often the case for HTTP (the WWW protocol). Over the last few years, the connection rate for a 'busy' web server has gone from dozens of connections per hour to dozens of connections per second. A strategy of "preforking" HTTP servers or using a single process and multiplexing requests with select() can result in a 10x-100x improvement in performance [1].

Configuration Errors

In the big picture, most of the easy performance gains arise from system configuration improvements. Most systems come out of the box with a configuration that aims to be a "jack of all trades" and, as it turns out, master of none. This means that you can greatly improve performance by examining and improving items like:

- basic kernel tuning,
- disk layout,
- system memory sizing and allocation,
- hardware selection and configuration,
- configuration of system services (e.g., a local DNS cache), and
- computer center scheduling.

Best case improvements can approach the 100x range.

Application Tuning

Usually, it is difficult to increase application performance by 10x or more. Out of the box, an application is likely to perform acceptably, though it might not be a stellar performer under heavy load. The first rule for applications is to choose your applications carefully and with an eye toward performance.

In most cases, vendor's code is not modifiable or tunable to a great extent. Be sure to take the time to understand how an application uses system resources and to investigate the tuning options that are available. For example:

- Should the application's files be split amoung several disk drives?
- Can unused features be switched off?
- Is the application logging too much data?

Kernel Tuning

Kernel algorithms must be well-matched to the task at hand. Basic kernel tuning is a fundamental tool in performance improvement. Regrettably, beyond basic tuning, few gains can be had without technological innovation (usually at great cost unless amortized across a very large number of systems).

Developing Expectations For System Performance

It is hard to tune a system without a basic collection of performance reference points to help you analyze the data provided by your monitoring tools. For example: File transfers typically run at between 650 and 750 KB/second on a active Ethernet at your site. Is this good? Could it be better? Or, a mail server you manage is handling 100,000 messages a day and seems to be saturated. Should you expect more? Where do you start looking for problems?

Having an "intuitive" feel for the answers to questions like these makes the process of performance analysis considerably more productive and more fun. Run your performance monitoring tools and get a feel for the numbers you see under various kinds of loads. In many cases, you can generate useful synthetic loads with a few lines of C or a bit of perl code. This is the information that gives you a feel for what to expect from your machines. It will save you from having to individually calculate and analyze each problem you take on.

Take the time to think through some of your system's basic limits. For example, what kind of performance can you expect from TCP/IP running on a 10 Mbit Ethernet?

Starting from the raw speed of 10,000,000 bits per second (note that 10,000,000 is 10 x 1000 x 1000, not 10 x 1024 x 1024), you have to calculate the bandwidth that will actually be available for carrying data (payload). First, calculate the link layer and protocol overheads:

8 bytes	Ethernet preamble and start of frame delimiter
14 bytes	Ethernet header
20 bytes	IP header
20 bytes	TCP header (assuming no options)
4 bytes	Ethernet CRC
12 bytes	Ethernet interframe gap
76 bytes	Total per packet overhead

Then measure (or estimate) the average packet size on the network you are interested in. You can use *netstat*(8) to get a count of the number of packets and bytes sent and received by an interface.

Next, make an estimate of the acknowledgement (ACK) overhead. If the traffic is predominantly in one direction, there will be ACK traffic that will reduce throughput. In the worst case, you would see one ACK for every two packets. Each ACK will consume 76 bytes unless it can be 'piggybacked' with data.

Pulling this information together into a table provides a picture of the performance to expect from TCP/IP on Ethernet.

Average Packet Size (bytes)	Uni-directional Traffic (KB/sec)	Bi-directional Traffic (KB/sec)
64	439	558
128	646	766
256	845	941
512	998	1063
1024	1098	1136
1500	1134	1162

Similar tables could be produced for UDP or any other protocol of interest.

The above numbers are an upper bound. A system might get close to these numbers during bulk transfers with little competing traffic. In an environment with several busy machines on the same wire, throughput will be somewhat lower, perhaps 70-90% of these numbers.

With these calculations in hand, you can try some experiments to see what happens in your environment. Most systems have rcp available, which offers an easy way to do a quick experiment. Contrary to what many folks think, ftp is usually not the best test of raw network capacity. Many ftp implementations use small buffers and are actually poor performers. Copy a big file (at least a megabyte) from one idle machine to another and keep track of the real-time spent. Do this two or three times to get an idea of your network's consistency. This experiment should reveal approximately the maximum throughput for your system as configured.

Try the experiment again on a loaded network. Note how badly (or not) the performance degrades in your environment.

It is worth doing this sort of analysis and experimentation for any subsystem you are interested in really understanding. Aaron Brown and Margo Seltzer presented a very interesting paper reporting their findings on the performance of Intel hardware at the 1997 USENIX conference [2].

Protocol Performance

Analysis and experimentation assist in understanding performance. Hard facts and knowledge about underlying algorithms can also contribute. Here are some interesting facts that contribute to overall performance of various network protocols:

- TCP/IP packets tend to be small, the average (according to one router vendor) is around 300 bytes.
- When a user's packets are being dropped (due to network load), network performance will be noticeably bad for that user (i.e., 20 second or more response time or 300 b/s FTP

throughput). This is because TCP backs off exponentially when retransmitting lost packets. This is good for the network as a whole, but bad for the affected connection. Good performance requires very low packet loss rates.

- HTTP and SMTP connections tend toward a small number of interactions that move a small or medium amount of data. For example, consider a WWW page that sets up and tears down 14 connections to load 14 small pictures (e.g., a bullet). This interaction pattern means that the TCP 'slow start algorithm' (which initially responds to requests a bit slower than possible to ensure that the network is not overwhelmed) will probably keep performance lower than is theoretically possible.
- New features often break things. While the design of TCP/IP supports adding new features without breaking existing implementations, it doesn't always work that way. It is worth keeping track of what's new on your network (and beyond), although the new code is not always where the bug lies. A common example of this phenomenon occurred a few years ago when TCP connections would fail when RFC1323 options were used. The problem was that some versions of Linux failed to handle the options correctly and corrupted packets. Fortunately, the folks implementing the support for RFC1323 had the forethought to implement an additional mechanism for turning them off on the fly. Another example concerned some Microsoft PPP implementations that would negotiate larger packet sizes than they could handle.

Understanding the Application

To improve overall performance, including applications, you're going to need to dig into the application and understand how it works and what it needs from the system.

Sadly, the application's documentation is usually not where to start. Tools like *ktrace*(1), *ps*(1), *top*(1), *netstat*(8), *tcpdump*(1) (or *etherfind*(1) in the Sun world) let you see how the application is interacting with your system and will yield real clues to the application's behavior.

INN (the news server) is an excellent example of an application that benefits greatly from study and understanding. Steve Hinkle's paper on INN tuning [3] is an excellent example of how to understand an application and then tune it for maximum performance.

Reference Points

As you build up a base of experience, you also build a set of reference points for what to expect from a system in real environments. These expectations can be really useful when trying to make initial "off the cuff" evaluations of system performance. For example, a good operating system can enable these levels of performance:

- A 90 MHz Pentium with 80 MB of RAM buffer cache can feed more than 3 million news articles (over 9GB) per day.
- A 150 MHz Pentium Pro can deliver over 1 million mail messages per day.
- A 133 MHz Pentium can serve static HTML from a cache at T3 speeds.
- A 150 MHz Pentium Pro can handle more than 3 million direct web hits per day, transferring 27 GB of content.
- A 133 MHz Pentium can only handle about 60 HTTP requests per second with a forking proxy.
- A 150 MHz Pentium running screend (user space packet filter) won't be able to handle a T1.
- A 266 MHz Pentium-II can't quite route a 100 Mbit Ethernet running at full load (full routing table, 256 byte packets). It can handle a T3.

These reference points aren't complete; they assume that you have the OS, hardware configuration, RAM, and other support (these reflect observations using BSD/OS 2.1 or 3.0, DEC or Intel Fast Ethernet Adapters, adequate memory and SCSI disks). Collect a set of reference points that are meaningful to you based on the systems that you are working with.

Collecting Data

Here are some hints for collecting performance data.

- **Get baseline data.** Without it you are lost, you won't be able to tell how much good (or bad) you have done.
- **Make sure you have enough.** For example, if you're trying to track down UDP packets "dropped due to no socket," make sure that your tcpdump has run for long enough that you expect to have collected several of these packets (*uptime*(1) and 'netstat -s' will help you pick an interval).
- **Make sure the data are representative.** Continuing with the above example, you also need to consider the possibility that the traffic you are interested in is not evenly distributed, and make sure that the data you collected represents an interval during which you could reasonably expect interesting traffic. To do this, you could start using *cron*(8) and netstat -s to sample every hour to determine when the traffic you are interested in occurs.
- **Make sure you sample frequently enough.** When you're looking for things like loading trends, you need to sample often enough to actually catch the event you're interested in. As a rule of thumb, you want to sample at at least twice the frequency of the event you are trying

to catch. If you don't know the frequency of the event, initially you may need to use a significantly higher sampling rate.

- **Automate your initial data reduction**. Tools like *tcpdump*(1) can generate huge amounts of data. Start by using tools like perl (and in the case of tcpdump, tcpdump's built-in filtering facility) to eliminate extraneous data. When you find no interesting data, don't forget to check your automated tools.
- **Check the log files**. Don't forget to check your system's log files and any logs files kept by the applications for clues.

Tuning the Subsystems

It is easier to look at the individual subsystems in relative isolation from each other.

The Disk Subsystem

Disk subsystem performance is often the single biggest performance problem on Unix systems. If your disks are busy at all (web, mail and news servers all qualify), this is the place to start.

Since the virtual memory system uses the disk when it is out of RAM, start by checking with *vmstat*(8) to ensure your system is not paging (the pages out 'po' column should be zero most of the time; it's OK to page in programs occasionally). If your system is paging, then skip the disk tuning for now and start by looking at RAM availability.

If your application has significant disk I/O requirements, the disk is almost certain to be your bottleneck. On a busy news or web server, correct disk configuration can easily improve performance by an order of magnitude or more.

Disk subsystem optimization has two main goals:

- minimize seek time, and
- match disk throughput to your system's demands.

When thinking about disk throughput, keep your system's load mix in mind. For some applications (e.g., streaming video), your system should maximize the rate at which data flows from the disk to the application. Rotational speed (the limiting factor on transfer rate) will likely dominate all other considerations. Other applications (e.g., mail and news servers) will see file creation and deletion operations dominate performance. File creation and deletion, which use synchronous operations to ensure data integrity, cause seek time to be the dominant factor in disk performance.

While modern SCSI disks are capable of transfer rates that approach or even exceed the speed of the SCSI bus (e.g., 10 MB/sec for Fast SCSI), the limiting factor for Unix systems is typically the number of file system operations that can be performed in a unit of time. For example, on a machine functioning as a mail relay, the limiting factor is going to be the speed of file creation and deletion. See Table 1 for a listing of SCSI speeds.

SCSI Version	MB/s
SCSI-I (async)	3.5
SCSI-I (sync)	5.0
Fast SCSI	10.0
Fast Wide SCSI	20.0
Ultra SCSI	20.0
Ultra Wide SCSI	40.0

Table 1: SCSI Bus Bandwidth.

In both cases, the only ways to increase performance (once you've hit the limit of the disk's physical characteristics) are:

- add disk spindles to increase parallelism,
- add hardware like PrestoServe to decrease seeks,
- identify and purchase a new file system technology that reduces seeks (test it carefully!)

In some cases it will be necessary to make changes to the application to make it aware of multiple directories or to use hardware that will transparently distribute the load across multiple spindles (e.g., RAID).

Disk Performance Factors

SCSI vs. IDE: For machines with heavy disk and/or CPU loads, SCSI is superior to IDE. A single system generally supports far larger numbers of SCSI disks than IDE; this can also be a consideration. With good host adapters, SCSI driver overhead is lower and 'disconnect' (the ability to issue a command to a drive and then 'disconnect' to issue a command to a different drive) is a big win. For machines that need only one or two disks and that have CPU cycles to spare, the lower cost of IDE is attractive.

Drive RPM: Rotational speed determines how fast the data moves under the heads, which places the upper bound on transfer speed. Today's really fast drives spin at 10,033 RPM (Seagate Cheetahs) and deliver about 15 MB/sec on the outside tracks. Last year's fast drives spun at 7200 RPM; inexpensive drives in use today spin in the 3600 to 5400 RPM range.

Average seek time: The average interval for the heads to move from one cylinder to another. Typically, this is quoted as about half of the maximum (inside track to outside track) seek time. Lower average seek times are very desirable. Currently, a 7-8 ms average seek time is really good, 8-10 ms is typical and much more than 10 ms starts to impact loaded system performance. Unix-like systems perform many seeks between the inode blocks and actual file data. To ensure file system integrity, writes of significant inode data are not cached, so these seeks are a significant part of the expense of operating a disk. In general, if a tradeoff must be made, a lower average seek time is better than a higher rotational speed.

The price of fast seek times and high RPMs is increased heat generation; take care to keep them cool or you will sacrifice performance for reliability. Use plenty of fans and make sure that there is good air flow through the box, or put the disks in their own enclosure(s).

On drive caches allow the drive to buffer data and therefore handle bursts of data in excess of media speed. Most modern drives have both read and write caches that can be enabled separately. The read cache speeds read operations by remembering the data passing under the heads before it is requested. A write cache, on the other hand, can cause problems if the drive claims that the data is on the disk before it is actually committed to the physical media (although some drives claim to be able to get the data to physical media even if power is lost while the data is still in the cache). Understand your hardware fully before enabling such a feature.

SCSI Tagged Command Queueing enables multiple commands to be issued serially to a single drive. Tagged command queueing increases disk drive performance in two ways: it enables the drive to optimize some mixes of commands overlap command decoding with command processing.

Avoiding Seeks

Some of best ways to improve performance involve reducing the number of operations to be performed. Since seek operations are performance eaters and relatively common, avoiding seeks can improve performance. Here are some methods to reduce the impact of disk seeks.

Use Multiple Spindles

Systems that concurrently access multiple files can improve their performance by putting those files on separate disk drives. This avoids the expense of seeks between files. Any busy server that maintains logs would do well to separate the log files from the server's data.

If you must store several Unix-style partitions on a large disk, try to put the most frequently used partition (swap or /usr) at the middle of the disk and the less frequently used ones (like home directories) at the outside edges.

Turn Off Access Time Updating

If maintaining a file's most recent access time (i.e., the time someone read it – not the time someone wrote it) is not critical to your site and, additionally, your data is predominantly read-only, save many seeks by disabling access time updating on the file system holding the data. This particularly benefits news servers. To turn off access time updates on BSD/OS, set the "noaccesstime" flag in the /etc/fstab file for the file system in question:

```
/dev/sp0a /var/news/spool
     ufs rw,noaccesstime 0
```

The same functionality exists in other Unix variants, but the names have been changed to confuse the innocent.

Use RAM Disk For /tmp

Many programs use the /tmp directory for temporary files and can benefit greatly from a faster /tmp implementation. On systems with adequate RAM, using 'RAM disk' or other in-memory file system can dramatically increase throughput. BSD/OS and other 4.4BSD derivatives support MFS, a memory file system that uses the swap device as backing store for data stored in memory. The following command creates a 16 MB MFS file system for /tmp:

```
mount -t mfs -o rw -s=32768 \
    /dev/sd0b /tmp
```

Or you could put this line in /etc/fstab:

```
/dev/sd0b /tmp mfs rw,-s=32768 0 0
```

The data stored on a MFS /tmp is lost if /tmp is unmounted or if the system crashes. Since /tmp is often cleared at reboot even when the file system is on disk, this does not seem to represent a significant problem.

Increasing Disk and SCSI Channel Throughput

Size The Buffer Cache Correctly

The best solution for disk bottleneck is to avoid disk operations altogether. The in-memory buffer cache tries to do this. On most 4.4BSD-derived systems, the buffer cache is allocated as a portion of the available RAM. By default, BSD/OS uses 10% of RAM. For some sites, this is not enough and should be increased. However, increasing the size of the buffer cache decreases the amount of memory available for user processes. If you increase the size of the buffer cache enough to force paging, all will be for naught.

Few systems support tools to report buffer cache utilization.

Use Multiple Disk Controllers

Adding extra disk controllers can increase throughput by allowing transfers to occur in parallel. Multiple disks on a single controller can cause bus contention, and thus lower performance, when both disks are ready to transfer data.

Striping (RAID 0)

RAID 0 distributes the contents of a file system among multiple drives. The result (when properly configured) is an increase in the bandwidth to the file system. The downside is that increasing the number of drives and controllers used to support a file system reduces the MTBF; RAID 0 does nothing to defend against hardware failures.

Higher Levels of RAID

RAID 1 (mirroring) provides the same write performance as RAID 0 at the expense of redundant disk drives (and, hence, higher cost per storage unit). Where performance is a major constraint and file system activity consists of many small writes (e-mail, news), RAID 1 may be the way to go. Of course, the redundancy can dramatically increase mean time to data loss.

Where read access dominates a file system's activity, RAID levels 3 (dedicated parity disk) and 5 (rotating parity disk) offer reliability and performance that is not significantly worse than RAID 0. RAID 3 and 5 offer reasonable write performance on large (relative to the stripe) files.

At the very high end, RAID hardware is implemented with a high degree of parallelism and sustained throughput can approach, or exceed, that of the system bus. Note, however, that write performance (particularly for smaller blocks) can be poor.

Monitoring and Tuning Disk Performance

The *iostat*(8) command can provide some help with optimizing disk performance. For each drive, it reports three statistics: sps (sectors transferred per second), tps (transfers per second), and msps (milliseconds per "seek," including rotational latency in addition to the time to position the heads). The accuracy of these numbers, especially msps, varies considerably among implementations.

The *tunefs*(8) command is used to modify the dynamic parameters of a file system. Disks whose contents are primarily read and that exploit read caching will often benefit from setting the rotational delay parameter to zero:

```
# tunefs -d 0 file system
```

When adjusting file system parameters with *tunefs*(8), remember that only new contents are affected by the change. Since the existing contents of the file system may constrain the allocation algorithms, it is best to use a scratch file system while experimenting with *tunefs*(8) options.

RAM

Two major performance factors related to RAM are:

- Avoiding paging (make sure that there is enough memory in the system to handle your load)
- Ensuring that enough memory has been allocated to the kernel.

Sizing Main Memory

If your system is short on main memory, it will end up swapping and the speed of your memory subsystem will degrade to the speed of your disk (usually about three orders of magnitude difference). The most useful tools for diagnosing paging problems are *vmstat*(8) and *pstat*(8). The 'po' column of *vmstat*(8) shows you 'page out' activity. Any value other than 0 on anything more than an occasional basis means that performance is suffering.

As you increase the amount of main memory, you are also increasing the likelihood of memory errors. The current crop of 64 MB SIMMs are especially prone to errors. You should plan on running Error Correcting Code (ECC) memory if your hardware supports it (and you should only be buying hardware that does!). Current ECC implementations on PC hardware cost about 5% in memory bandwidth when accessing main memory (and the CPU tends to go to main memory less frequently than one might expect).

When sizing a system's RAM, it is often useful to know the memory usage of a typical user or a particular process. On BSD derivatives, you can get some of this information with 'ps -avx.'

The RSS ('resident set size') column reports the number of kilobytes in RAM for a process. Since the shared libraries and the text are common to all of the instances of a program (and in the case of the shared libraries to all of the users of the library), you can't use this information to determine the amount of memory that would consumed by an additional instance of the program. It does give an upper bound for making estimates.

The TSZ ('text size') column reports the size of the program's text, but it does not tell you how much of the text is resident (i.e., how many pages are not in the swap area).

After observing these parameters for a while, you can get a pretty good idea of how much memory a particular process typically uses. It may also be worth watching the numbers while you subject a process to both typical and extraordinary loads. Using this information, you can then estimate how much memory you would need to support a particular mix of applications. Perl and cron can help to automate this task.

Cache Size

Currently, tools don't exist to directly monitor real life cache performance. Within reason, more cache is better. Run real-life benchmarks to see if different hardware can improve your site's performance.

Kernel Memory Tuning

Most modern kernels can be tuned for increased loads using only a "knob" or two, but for some applications you will need to do additional tuning to get peak performance.

On BSD/OS, the size of kernel memory has a default upper bound of 248 MB. Major components of kernel memory are the buffer cache (BUFMEM) and the mbufclusters (NMBCLUSTERS). Both are in addition to the memory allocated by KMEMSIZE. On most systems, these memory allocations are completely adequate, but if the size of the buffer cache is increased beyond 128 MB it may get tight.

The limit on kernel memory can be increased by modifying the include file machine/vmlayout.h (with suitable knowledge of the processor architecture in question) and then recompiling the entire kernel. In addition to the kernel, you'll also need to recompile libkvm (both the shared and static versions), gdb, ps and probably a few other programs.

maxusers

On BSD systems of recent vintage, the place to start kernel memory tuning is with the 'maxusers' configuration parameter. The maxusers parameter is the "knob" by which the kernel resources are scaled to differing loads. Don't think of this in terms of actual numbers of humans, but just as a knob that can request "more" or "less."

As a rough rule of thumb, you can start by setting maxusers to the size of main memory (e.g., for a system with 64 MB of main memory start by setting maxusers to 64).

maxbufmem

The kernel variable maxbufmem is used to size the buffer cache in systems without a unified user space/buffer cache. A zero value means "use the default amount (10% of RAM)." Otherwise, it is set to the number of bytes of memory to allocate to the cache. You can set this at compile time:

```
options "BUFMEM=\(32*1024*1024\)"
```

Or you can patch the kernel image (with *bpatch*(1) or a similar tool):

```
# bpatch -l maxbufmem 33554432
```

and reboot. It is not safe to change the size of the buffer cache in a running system.

A system running a very busy news or web server is an obvious candidate for increasing the size of the buffer cache.

When increasing the size of the buffer cache, the memory available for user processes is decreased. Ideally, a tool would report buffer cache utilization. Unfortunately, such a tool doesn't seem to exist, so tuning is sort of hit-or-miss – increase the size of the buffer cache until you start paging, then back off.

mbufs and NMBCLUSTERS

In the BSD networking implementation, network memory is allocated in mbufs and mbuf clusters. An mbuf is small (128 bytes). When an object requires more than a couple of mbufs, it is stored in an mbuf cluster referred to by the mbuf. The size of an mbuf cluster (MCLBYTES) can vary by processor architecture; on Intel processors, it is 2048 bytes.

The number of mbufs in the system is controlled by their type allocation limit (reported by vmstat -m). The configuration option NMBCLUSTERS is used to set the number of mbuf clusters allocated for the network. The default value for NMBCLUSTERS in the kernel is 256. If you have GATEWAY enabled, NMBCLUSTERS is increased to 512. Systems that are network I/O intensive, such as web servers, might want to increase this to 2048.

On recent BSD systems this parameter can be dynamically tuned with *sysctl*(8):

```
# sysctl -w net.socket.nmbclusters=512
```

The upper bound on the number of mbuf clusters is set by MAXMBCLUSTERS. Usually MAXMBCLUSTERS is set to 0 and the limit is calculated dynamically at boot time.

If a BSD kernel runs out of mbuf clusters, the kernel will log a message ("mb_map full, NMBCLUSTERS (%d) too small?") and resort to using mbufs. This can also be seen by an increase in the number of mbufs reported by vmstat -m.

KMEMSIZE

The size of kernel memory (aside from the buffer cache and mbuf clusters) is set by KMEMSIZE. Normally, it is scaled from "maxusers" and the amount of memory on the system, but it can also be set directly.

The default KMEMSIZE is 2 MB. At a maxusers value of 64 (or if there is more than 64 MB of memory on the system), KMEMSIZE will increase to 4MB. At a maxusers of 128, KMEMSIZE will increase to 8MB. Beyond that, use the KMEMSIZE option (in the kernel configuration file) to increase the kernel size.

Routing Tables

To run a full Internet routing table in the Spring of 1997, it is necessary to increase the amount of kernel memory to at least 16 MB. The BSD/OS config file for the GENERIC has notes on this:

```
# support for large routing tables,
# e.g., gated with full Internet
# routing:
# options "KMEMSIZE=\(16*1024*1024\)"
# options "DFLDSIZ=\(32*1024*1024\)"
# options "DFLSSIZ=\(4*1024*1024\)"
```

Since the gated process is also likely to get quite large, it also makes sense to increase the default process data and stack sizes.

The *vmstat*(8) can give an overwhelming amount of information about kernel memory resources. Some points worth noting: In the "Memory statistics by type" section, the "Type Limit" and "Kern Limit" columns report the number of times the kernel has hit a limit on memory consumption. Entries that are non-zero bear some investigation. These statistics are collected since system boot, so you'll probably want to look at the difference between two runs that bracket an interesting load. Many of these limits scale with maxusers, so a quick experiment can be done by recompiling the kernel with a higher maxusers value.

Network Performance

For network performance analysis, *netstat*(8) is the command you want. The raw output of a single *netstat*(8) run is not terribly useful, since its counters are initialized at boot time. Save the output in a file and use diff or some fancy perl script to show you the changes over an interval.

For understanding why your network performance sucks, *tcpdump*(1) or some other sniffer/protocol analyzer is the tool of choice. Data collected with *tcpdump*(1) can be massaged with perl (or sed and awk) to make it easier to see what is going on.

Too Much Traffic

Take time to check that you've only got the traffic you expect on the network. Look for things like DNS traffic (if you have much, configure a local cache), miscellaneous daemons you don't need (not only are they costing you network traffic, they are probably costing you context switches), etc. If you're really trying to squeeze the last little bit out of the wire, consider using multi-cast for things like time synchronization. Use *tcpdump*(1) to see what kind of traffic there is. Watch for unexpected activity, or unexpected amounts of activity, on your hubs. Compare interface statistics with similar machines.

Too Little Bandwidth

With Ethernet, as the amount of traffic on the network increases, the instantaneous availability of the network decreases as the hardware experiences collisions. Use 'netstat -I' to monitor the number of collisions as a percentage of the traffic on the wire. Ethernet switches can reduce the number of hardware collisions and increase apparent number of segments. Full duplex support (e.g., 10baseT) can also help.

Errors

Significant error rates (anything more than a fraction of a percent) are often an indication of hardware problems (a bad cable, a failing interface, a missing terminator, etc.).

Use 'netstat -I' to monitor the error rate.

Timeouts and Retransmissions

Don't forget to watch the activity lights on your interfaces, hubs, and switches. Look for unexpected traffic, pauses, etc. and then get busy with *tcpdump*(1) to see what is going on. This paper will cover some very basic concepts; for more detail, see Richard Steven's excellent book, *TCP/IP Illustrated, Volume 1* [4].

Buffer Sizes

Network buffer sizes can have a significant impact on performance. With FDDI, for example, it is often necessary to increase the TCP send and receive space in order to get acceptable performance.

On BSD/OS and other 4.4 BSD derivatives this can be done with *sysctl*(8); see Listing 1. To a first approximation, think of net.inet.tcp.recvspace as controlling the size of the window advertised by TCP. The net.inet.tcp.sendspace variable controls the amount of data that the kernel will buffer for a user process.

As a general rule of thumb, start sizing the kernel buffers to provide room for five or six packets "in flight" (one packet for the sender to be working on, one packet on the wire, one packet being processed at the receiver – then multiply by two to account for the returning acknowledgements). Since Ethernet packets are 1500 bytes (or less), 8 KB of buffer space is about right. For FDDI, with 4 KB packets, 20 to 24 KB is likely to be necessary to get FDDI performance to live up to its potential.

Another way to think about this is in terms of a "bandwidth delay product:" multiply bandwidth times the round trip time (RTT) for a packet and specify buffering for that much data. If you want to be able to maintain performance in the face of occasional lost packets, figure the bandwidth delay product as:

buffer bytes = bandwidth bytes/second *
(1 + N packets) * RTT seconds

Where N is the number of lost packets you want to be able to sustain without losing performance. This is a bit harder to calculate as you have to be able to measure RTT.

Increasing the size of the kernel send buffers can improve the performance of applications that write to the network, but it can also deplete kernel resources when the data is delivered slowly (since it is being buffered by the kernel instead of the application).

The size of application buffers should also be considered. If the buffers are too small, considerable additional system call overhead can be incurred.

Using netstat(8)

netstat -I

Using 'netstat -I' yields a basic report on the amount of traffic and the number of errors and collisions seen by a network interface. Here are some of the statistics reported:

- **Input Errors** tells you that a bad packet was received – something is wrong somewhere between you and the sender.
- **Output Errors** means something is wrong with the local hardware (interface card, cables, etc.).

```
# sysctl -w net.inet.tcp.sendspace = 24576
# sysctl -w net.inet.tcp.recvspace = 24576
```

Listing 1: Changing network buffer sizes.

If your hardware supports full duplex operation, another possibility is that one end is configured for full duplex while the other is not.

- **Collisions** tells you how busy the network segment you're attached to is. Lots of collisions (say more than 10%) on a regular basis tell you it's time to think about reworking your network architecture.

Another use of 'netstat -I' is tracking down lost packets. Say you are trying to ping a remote machine and are not receiving any replies. You can track down the location of the problem by observing the input and output packet counts of the machines along the way. Each machine that processes the packet will increment the appropriate counters. On the machine where the packet was "lost" you would expect to see an increasing error count, either on input or output.

netstat -s -I

Specifying 'netstat -s -I' yields more detail on the traffic seen by the interface, including the number of dropped packets, the number of octets (bytes) sent and received (which, along with the packet count, enables you to compute average packet size) and data on multicast traffic.

netstat -s

With 'netstat -s,' you get voluminous statistics on the whole networking subsystem. These statistics are maintained over the life of the system, so to see current trends, look at a pair of reports and compare the numbers. A quick and dirty way to do this is to save the output into a pair of files and use diff. Of particular interest are the counts of:

- **Dropped packets** indicates that the local machine is receiving network traffic but does not have the resources to handle that traffic. The two most common causes are: running out of mbufs or running out of CPU power. Another possibility is that you've run out of mbuf clusters and the kernel is substituting mbufs, but the system is marginal on CPU power and the extra processing involved has pushed it over the edge.
- **Fast Retransmits**. The TCP spec requires that an immediate ACK be sent when a packet is received out of order. This ACK will duplicate an ACK that has already been sent for the last in-order data that the receiver received. When the sender has seen three duplicate ACKs, it assumes that the next packet has been lost and it will retransmit the packet without waiting for a retransmit timeout. This is a "fast retransmit" and, if the receiver's window is big enough, it will prevent TCP from waiting for a timeout when a packet is lost.
- **Duplicates**. Duplicate packets are packets that arrived twice. Usually this means that an acknowledgement was delayed and the sender retransmitted.
- **Retransmit Timeouts**. The retransmit timer expired while we were waiting for an ACK.
- **IP Bad Header Checksums and TCP Discarded For Bad Checksums**. Packets that were "damaged in transit" should be thrown out by the link layer checksum and not seen by any of the higher layers. This is a possible sign that an intermediate gateway is corrupting packets. If you are seeing significant numbers of these, it might be worth the trouble to try to track down the source. To do this, you will need to probe methodically, starting with the nearest gateways (routers) and working outward.
- **UDP with no checksum**. These are interesting because running UDP without checksums is asking for trouble. If you are seeing many of these, it would be worth tracking down the source.
- **UDP with no socket** means that you're getting UDP traffic, but nobody is listening for it. At the very least, this is a waste of resources, and it may be indicative of a configuration error or perhaps even something malicious. Track the source using *tcpdump*(8) if many packets show up here.

When the network is performing well, all of the above statistics should represent a small fraction of the traffic that the system sees (well under a fraction of a percent).

netstat -a

With 'netstat -a,' you get information on all of the active network connections on the system. Things of interest include:

- **The number of connections.** This gives you a feel for the amount of network traffic the system is seeing.
- **Connections with with data in the Send-Q or Recv-Q**. You will see this occasionally, but significant numbers of connections with queued data would indicate that something is amiss.
- **Large numbers of connections in states other than ESTABLISHED.** On a busy web server, many connections will be in the TIME_WAIT state. But, observing significant numbers of connections in SYN_SENT (indicative of a SYN-flood attack) or FIN_WAIT_2 (waiting for a FIN from the other side after we've sent our FIN, sometimes this indicates a kernel bug) bear investigation.

netstat -m

With 'netstat -m,' you get a report on the memory usage of the networking subsystem. When investigating performance problems, the counts of "requests for memory denied/delayed" and of "calls to the protocol drain routines" are of particular interest. They are indications that the network is running low (or out) of memory.

Using tcpdump(1)

The *tcpdump*(1) program is a powerful tool for analyzing network behavior and is available on most Unix-like systems. With *tcpdump*(1), you can watch the network traffic between two machines. The packet contents are printed out in a semi-readable form (which is easy to massage with perl to help you see the info you're interested in). Each packet is printed, along with a time stamp, on a separate line.

One of the most useful features of tcpdump is the ability to filter traffic. For example, if you're curious about stray traffic on the network with your web servers, you could run something like this:

```
# tcpdump -i de0 not port http
```

If you had connected to the machine by telnet you might use:

```
# tcpdump -i de0 not port http \
    and not port telnet
```

The output from tcpdump would only show non-http and non-telnet traffic, thus enabling you to focus on the traffic that you're interested in.

You can save the data for later analysis by having tcpdump write it to a file:

```
# tcpdump -i de0 -w tcpdump.out
```

When you examine the data later, you'd use:

```
# tcpdump -r tcpdump.out not \
  port http and not port telnet
```

Aside from making analysis of the data more flexible, this approach also consumes fewer machine resources, so you're less likely to drop packets or otherwise impact the system under test. By default, tcpdump only grabs the first 76 bytes of the packet (which is enough to get the headers, but you won't get much of the packet content). You can add '-s 1600' to save the whole packet.

One of the most common things that you'll be looking for is an explanation for why the network is pausing every so often.

Here is a recipe for examining tcpdump output:

1. Collect the data, and post-process it if you want to (for example, you may want to convert the time stamps into relative intervals).
2. Browse through the tcpdump output with your favorite editor, looking for places where the timestamp jumps by more than the "usual" amount.
3. You're probably in the vicinity of a dropped packet, and the timestamp is due to the timeout before the packet was retransmitted.
4. Take a quick look around and then start looking for the next timestamp jump. What you're looking for is a pattern. If the network behavior was bad enough to get you looking at packet traces there is almost certain to be one.
5. After some looking, a pattern will emerge. If you don't know what the problem is by now, you'll need to find somebody who knows more about the protocol in question. But, armed with a description of the problem, the tcpdump output, and the patterns you found, it should be easier to get some help.
6. As you're looking at the jumps in the tcpdump output, an important part of the story is the size of the jumps. Are they the same? Are they increasing? There are people who know this stuff well enough to make a good guess as to the source of the problem just by the size of the delay.

In some cases, you will need to arrange for the machine running *tcpdump*(1) to be in the middle of the wire between the machines that are having problems. With some switches this is not possible, and you will either need to substitute a hub (you can also just add a hub at one of the switch ports and connect your monitor and one of the machines under test to the hub) or run tcpdump on the machines at both ends of the link and analyze both sets of data. The hub trick doesn't work if you're trying to look at a link that is running full duplex.

CPU

In performance analysis, you're usually up against one of two things: either CPU utilization is too high or else it's too low. Too high means that you've got to figure out a way to free up more cycles in order to get more work done. Too low means that you've got to figure out what you're waiting on, usually some I/O device.

Start CPU analysis with a tool that will tell you how your CPU is being utilized: *top*(1), *vmstat*(8),) and *iostat*(8) will all tell you how much of your CPU is being used and whether the cycles are being spent in user or kernel (system) code. If most of the CPU's time is being spent executing user code, *top*(1) and *ps*(1) will help you identify the processes of interest. If your CPU is close to 100% utilized, you need to figure out what it's doing. Start by identifying the process or processes that is using most of the user time.

Even if the CPU is spending most of its time executing in the kernel (high percentage of system time), these processes are likely to be the cause. Once the processes in question have been identified, *ktrace*(1) can be a very useful tool for figuring out what is going on. So are the 'in' (interrupts), 'sy' (system calls) and 'cs' (context switches) columns in the output of *vmstat*(8). You should be developing a rough idea of the performance bounds of the systems you support. This is the information you will need to read the output of commands like *vmstat*(8).

Very simple benchmarks (say 10 lines of code) can help you calibrate your expectations of the machine's performance. For example: A 200 MHz P6 can perform about 500 fork/exit/wait (i.e., null fork

with two context switches) operations per second. It can also perform about 400,000 getpid() operations per second (getpid() is arguably the most trivial system call). From these benchmarks, you can make an educated guess that a process that is fork()ing 10s of times per second should not be putting too much of a load on the system due to the forks, but a process that is doing 100s of fork()s per second may well benefit from a redesign.

The *ktrace*(1) program can give insight into the behavior of the processes at the system call level. Since system calls are relatively expensive, this can be very useful. Often the information revealed by ktrace will be a surprise, revealing details of the program's operation that were neither documented nor intuitive.

For example, some (maybe all) versions of Apache search from the root for a .htaccess file. If you're not using these files, turning off the check can get you a 5 to 10% increase in performance. If you are using them, you can decide if it would be worth the trouble to use a "shallower" directory hierarchy. This is also a great way to figure out error messages (like "file not found" with no file name or without a full path).

Other things to look for are frequent read() and write() calls or such calls with small byte counts, frequent system calls of any type (perhaps the application can cache the data), etc.

If all else fails, you may have to resort to profiling your application or your kernel. Typically this means that you have to either have source code or vendor cooperation. Enabling profiling usually means you need to recompile. You can get clues to the application architecture, and determine if the problem lies in the application or in the system libraries.

If the problem appears to be in the kernel, profiling can be a very useful tool. Even if you don't do kernel work, useful information can be gained. For example, if you discover that a significant amount of time is being spent in a device driver, you might want to try a different device. If you have a cooperative OS vendor, being able to tell them the location of your kernel's hot spots might be just the incentive they need to work on that particular code.

The *time*(1) command can help you distinguish between CPU under-utilization due to outside delays (disk or network) and having an excess of CPU capacity. When time reports a wall clock (real) time that is significantly longer than the combined user and system times, you should suspect that you're seeing an I/O (or memory) problem. Small variances are to be expected due to other processes being given time to run.

Some types of loads will result in misleading numbers from *vmstat*(8) and its friends. For example, on a gateway, the load of handling the network interfaces will not be reported by *top*(1) or *vmstat*(8). In such cases a "cycle soaker" can be a useful thing. It will tell you how much CPU is actually available for computation.

Profiling User Code

If most of your CPU time is being spent in user mode and *ktrace*(1) doesn't give you enough information to improve the situation (or to place the blame and move on), then profiling the program in question might be the way to go. To profile the application you (or the vendor) will have to recompile the code. If this is not possible, you will need to choose another strategy.

The procedure for profiling user code is slightly different than that for profiling the kernel:

- Arrange to run the C compiler with the -pg flag. Probably the easiest way to do this is by adding it to the CFLAGS in the Makefile for the program. You may also need to arrange to link statically.
- Run the program. When it terminates, the file gmon.out will be produced in the directory where program was run. You won't get gmon.out until the program terminates.
- Display the profiling data with *gprof*(1). The profiling data will not include kernel time spent on the process's behalf. It may be necessary to do a bit of piecing together evidence from kernel and user profiling in order to get a full picture of where your cycles are going.

One of the challenges of profiling servers is to get the server to run in a mode where you can collect useful data. Servers that fork to handle requests will have one instance of the server process that will typically run for the life of the system, forking a new instance each time a request comes in. In this case, two different threads of execution will need to be profiled. The easy one to profile is the "master" instance that accepts the incoming requests and then forks to handle them. The processes created to handle the requests are harder to profile, and due to their short life may not produce statistically valid data. To get around this you may have to figure out a way to get the server to handle requests directly without forking.

Profiling the kernel

Typically, this is only interesting when you're out of CPU and want to figure out where the cycles went **and** you're really sure that the kernel is at fault. To do this, you'll need to build a profiling kernel and then reboot with that kernel. Profiling can be turned on and off, allowing you to collect data representative of the load you are interested in. Running a profiling kernel does add a bit of overhead to the system.

Here are the steps for doing this on BSD/OS:

- Configure and compile a profiling kernel:

```
# cd /usr/src/sys/i386/conf
# config -p MY_KERNEL
# cd ../../compile/MY_KERNEL.PROF
```

```
# make depend all
```

- Install the new kernel and reboot.
- When you are ready to collect profiling data, enable profiling with:

```
# kgmon -rb
```

- Run your load.
- When you're done collecting data, disable profiling with:

```
# kgmon -h
```

- Then dump the collected data with:

```
kgmon -p
```

  The data is left in a file called gmon.out in the current directory.
- The profiling data is display by *gprof*(1). gprof has almost as many options as ls, but the following should work pretty well for starters:

```
# gprof /bsd gmon.out
```

  You may want to redirect output to a file so that you can use perl to help you make sense of the data.

For more details, take a look at the *config*(8), *kgmon*(8) and *gprof*(1) man pages.

Typically, what you're looking for in the profiling data is a rough idea of where the kernel is spending its time. Is it a device driver (which one)? The file system? The network, etc.? Often the answer is surprising. What you do next depends on where the data leads you and the amount of kernel hacking you want (or are allowed) to do.

For example, if the data points to the Ethernet driver (as in the first example below), you will either want to experiment with a different card or dig deeper into the profiling data to see if you can find some room for improvement in the driver (or suggest that your OS or Ethernet vendor do the same). On the other hand, if the data points you to fork(), the place to look is at your load.

Kernel Profiling Example

See Figure 1 for sample kernel profiling output. The data is from a profiling run on my laptop with a heavy load of network traffic. The Ethernet interface is a 3com 3c589B which uses the ef driver.

Looking at the data, one of the first things to notice is that the efintr() routine is right up near the top, so we can pretty reasonably focus our interest on the Ethernet interface. One of the next things to notice is that most of the time in efintr() is attributed to insw(). insw() is part of the machine dependent assembler code in locore.s, doing signed 16 bit reads from the Ethernet interface. This is not surprising; since the 3c589B uses Programmed I/O (PIO), it is also a reasonable bet that insw() was written with considerable concern for efficiency. In this case, it looks like the right thing to do is to try a different Ethernet card, looking for one that does DMA. NH Code Availability

A collection of useful benchmarks (and pointers to more) can be found at ftp://ftp.bsdi.com/bsdi/benchmarks. Source code for tcpdump can be found at ftp.ee.lbl.gov.

```
                                called/total     parents
index  %time    self descendants  called+self    name           index
called/total     children
                                                <spontaneous> [1]     53.0
0.19     85.75                              doreti [1]
                 0.19     80.64            97968/97968      _isa_intrswitch [2]
                 0.76     4.02             22237/43750      _ipintr [11]
                 0.00     0.14              2264/3158       _softclock [41]
                 0.00     0.00                60/971        _trap [53]
                 0.00     0.00                 1/1 _arpintr [403]
-----------------------------------------------
                 0.19     80.64            97968/97968      doreti [1]
[2]      49.9    0.19     80.64            97968          _isa_intrswitch [2]
                 25.63    54.99            97893/97893      _efintr [3]
                 0.00     0.01                75/75         _wdcintr [91]
-----------------------------------------------
                 25.63    54.99            97893/97893      _isa_intrswitch [2]
[3]      49.8    25.63    54.99            97893          _efintr [3]
                 53.68    0.00 12626901/12626907     _insw [5]
                 0.74     0.00             97893/97893      _ef_newm [24]
                 0.56     0.01             97893/97893      _ether_input [27]
```

Figure 1: Kernel profiling output.

Author Information

Douglas Urner has been working with Unix-like systems for 15 years. He was a project leader in the Small Systems Support Group at Tektronix (when a VAX 780 was a "small" system). Today he is a member of the technical staff at Berkeley Software Design, Inc. (BSDI) where he mediates the battle between good and evil (or is that engineering and sales?).

References

[1] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz and Kurt J. Worrell. *A Hierarchical Internet Object Cache.* Technical Report 95-611, Computer Science Department, University of Southern California, Los Angeles, California, March 1995. Also Technical Report CU-CS-766-95, Department of Computer Science, University of Colorado, Boulder, Colorado, http://harvest.transarc.com/afs/transarc.com/public/trg/Harvest/papers.html#cache .

[2] Aaron Brown and Margo Seltzer, "Operating System Benchmarking in the Wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture," *Invited Talk Notes of the 1997 USENIX Conference,* Anaheim, CA, http://www.eecs.harvard.edu/vino/perf/hbench.htm .

[3] Steven Hinkle, "Tuning INN News on BSD/OS 3.0," http://www.bsdi. com/ .

[4] W. Richard Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison Wesley, 1994.

Automation of Site Configuration Management

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

Although there are countless tools to track and manage the configuration of large numbers of Unix systems, there seems to be a lack of tools to manage the interaction and dependencies between systems. As our site has grown, many machines provide services that are required for the operation of other machines and applications. We have been unable to maintain accurate lists of services and servers, and even routine system upgrades have resulted in unexpected service outages.

To address this problem, we are developing a system to automatically detect many of these service dependencies, and generate up to date server listings. In addition, it provides a general framework for indexing and accessing troubleshooting, operational, installation and a number of other types of documentation. The system also assists in verifying the configuration of systems being installed, and assists with the real time testing of services.

Introduction

As the number of Unix machines being supported grows, support groups develop or install more and more tools to automate and streamline the system installation and update process. These range from moderately simple file copying tools such as rdist or package[1] to commercial products such as "Tivoli," to any number of individual development efforts such as *Config* [12], or GeNUAdmin [9] and others that can be found in the proceedings from past LISA conferences. Like these sites, we have also developed and installed tools to manage system configuration.

In addition to managing system configuration, it is often useful to collect system configuration (both hardware and software) and state information in a central location. This can be done in a number of ways, such as with syslog [14] or SNMP [11] or via the file system (see the section on *My-State*).

While we would all like to work in a problem free environment, that is often not possible. Failing that, it is desirable to at least detect system problems before our users do. Since services are often spread over many machines, tools have been developed to detect problems on these machines, either by waiting for error messages with tools such as *swatch93* [8], or by going out and testing things with tools like *pong* [10].

While all of these tools fill important roles in managing large numbers of workstations, where they are generally lacking is in dealing with the dependencies **between** the applications and servers, and with the maintenance of the configuration files. For example, while *pong* is a great tool for testing large numbers of services, it can't test servers that no one told it about. While there has been some work in the area of detecting these dependencies, these have either been limited in scope to the network configuration such as *Fremont* [15] or *Ined* [13], or have it as part of general configuration management system, and is restricted to checking just what is managed via that system [1].

An often repeated request from our Network Operation Center (NOC), was an up to date version of the *Server List*. This was a list of all of our server machines, and the services each one provided. These services ranged from domain name service, to print servers, to license servers (with multiple applications per server), authentication servers, time servers, mail servers, file servers, PC and Mac servers and so on. To make matters worse, there were often multiple services per machine, and services were moving from one machine to another machine to accommodate changes in OS level, changes in network topology, security and performance concerns. In addition, short term test services would often be set up in odd places to provide a new service, or attempt to isolate a problem with some other service, and be left in operation accidentally. This has resulted in a number of unplanned service degradations or complete failures, when one staff member takes a machine down for service or redeployment, not knowing that someone else had set up some special service on it. The most recent example, is a week after the successful upgrade of two DNS and license service machine, we remembered that the machine was also the YP master for our site. It wasn't until we created a batch of userids for a special class and no one could sign on, that we discovered the

[1]Package is a tool supplied with AFS that *pulls* in files and directories to a machine based on a configuration file. It detects new versions and installs them as needed.

problem. The YP master function has been moved there a year ago when the real YP master machine had a CPU failure, and never got moved back. Oops.

Since this was not the first time something like this happened, and that the request from the NOC was quite reasonable, we decided it was time to come up with a solution. For the past six years, the RCS[2] userids at RPI have been managed with a locally developed package called Simon [2, 3] which is built on top of an Oracle relational database. In addition to managing userids, this system also managed other aspects of our site such as the printing configuration file [4], the host table and DNS files, and even such mundane things as our telephone directory [6].

Given our past success with using Simon (and Oracle) to solve any problem, the direction for this solution became pretty clear. While the request from the NOC was for a "server list," given the nature of the data, a hypertext document would be better for general use as you navigate from service to server instance to client and back. Between the telephone directory project, and other projects that required the generation of HTML pages from the database [5], we had the underlying technology to approach the problem.

MyState

Like many other sites, we had the desire to be able to track system configuration in a central location. To this end, we wrote a *Self_Exam* script that maintains a file called `/etc/MyState` which holds a number of records with information on that particular system. This file would then be moved back into our central file system as part of the standard *Post_Package*[3] run and placed in a `ShippedBackFiles/MyState` directory, using the hostname as the filename.

Each record in the file consisted of three fields, the item name, the item value, and the method used to obtain the value. This was important as different OS versions would report the same item in different ways or in different units. The data from a typical `/etc/MyState` file is shown in Table 1.

These `MyState` files were useful just as stand alone files. Since we had them all in the central file-system, all stored in the same directory, you could make quick searches for things by using simple UNIX tools such as `grep *`, or just simply `cat` out the file for the host of interest. Unfortunately, that was about all you could do with it. Other information such as the location of the machine, the owner, maintenance status, and so forth was not readily available. In addition, we often were interested in grouping machines by different attributes. For example, OS upgrades often required that we identify all machines of a certain type, with less than a certain amount of disk or ram. In addition, there was no facility for tracking changes in configuration, or even knowing when a particular file was obtained (knowing when the configuration of a machine changed is handy when you have memory thieves operating at your site!).

In addition to the `MyState` files, there are often other interesting machine configuration available on some platforms. For instance, under AIX, you can issue an `lscfg` command and get lots of useful information about the hardware configuration of a machine. We often find ourselves juggling machine configurations, moving interface cards, disks and memory between machines as our needs changes. While the `MyState` file could tell you how much RAM a machine had, it would not tell you if you had one 32MB SIMM or eight 4MB SIMMs installed in a machine (and how many available RAM slots were left). Rather than visiting every machine when we

[2]RCS, the Rensselaer Computing System, a collection of 700 workstations and Unix timesharing machines available to all students, faculty and staff.

[3]After a successful package run, a post package script was run that would restart servers, and clean up installation details that can not be handled by simply copying in a file.

Name	Value	Method
hostname	simonsrv.sss.rpi.edu	hostname
hostid	0x8071640c	hostid
ipaddress	128.113.100.12	grepped from /etc/hosts
gateway	128.113.100.244	netstat -r
ypmaster	asher.its.rpi.edu	ypwhich
memory	128M	lscfg
macaddress	08.00.5a.cd.5d.42	netstat
arch	power	assigned to all rs6000s
model	rs6000 250	uname -a
os	AIX 4.1	uname -a
swap	256M	lsps -a
disk	5596M	lspv

Table 1: Sample `/etc/MyState` file contents

wanted to ask these questions, or expanding or duplicating the existing `MyState` files, we decided to dump everything into the database.

To this end we defined a new table, `System_State`, in the Simon database (see Table 2). We make an entry in the table for each line in each `MyState` file. When we first load a `MyState` file, we record the `System_Id` [4] of the host, the means used to collect the data (in this case, `MyState`), the source of the data (in this case, the `MyState` file), and for each line, the name of the item, the current value of the item, and the third field. We also record the current time and date in the `Date_Obtained` field.

When we run the `load_system_state` program on a machine, it selects all the existing records for that particular machine and means, and sorts them by `Item_Name`. It then reads and sorts the data from the `MyState` file. Once both of these lists are in place, it compares them one by one. If there is a new entry, it is inserted as described above. If an entry is missing, the record is marked as deleted by setting the `Date_Obsolete` field to the current date. if the `Item_Names` match, the `Item_Value` and `Item_Extra` are compared. If they are still the same, the `Date_Verified` field is updated. If they are different, the old record is deleted and a new record is inserted with the updated information.

In order to handle data sources besides `/etc/mystate`, the `load_system_state` program has a list of items to check. Each item has either a filename to read or a pathname to execute, a flag indicating file or program, a standard value for the `Means`, some parsing information[5] to assist in decoding the information and what operating system the entry can handle. The OS information lets us collect information using vendor specific commands, such as `lscfg` under AIX.

All of this information is available to other database applications. For instance, we will very likely use the "OS" information for a machine to sort the application usage via license server logs [7]. This will be useful in deciding how to handle version differences of applications across the different platforms. Of perhaps more general use to our staff, we also generate web pages, one for each system, that include administrative information (owner, sys admin, contract status, etc), as well as the information for that system stored in the `System_State` table. We also generate index pages for each of the common items and common values. When generating these pages, the extraction program attempts to identify common attributes (such as memory or disk size) and generate the page. Unique

[4] The `System_Id` is an internal database key that corresponds to one particular machine. One advantage of this, is that the entries (except for hostname!) are not impacted by a host name or domain change, and we can trace the history of a physical machine.

[5] Currently we support fixed columns for the fields or character delimited fields.

Name	Type	Size	Description
System_Id	Num	22	The Simon.Systems.System_Id of the machine in question.
Means	Char	12	The facility used to obtain this information. This might be /etc/My_State, or possible platform specific configuration commands such as lscfg. All hosts should have at least MyState info recorded.
Item_Name	Char	32	The name of the particular item we are recording. For MyState files, this would be the first column. This essentially addresses the question "what"
Item_Value	Char	256	The value for this particular item. For MyState, this is the second column. This tells us "how much."
Item_Extra	Char	256	An additional field to handle extra info for this record. For the MyState file, this is the means to obtain this particular bit of information. This can be important when attempting to compare the same "what," since different units or methodologies may have been employed to get the result. The specific definition of this field depends on the means.
Date_Obtained	Date	7	The date when we first encountered a record of this type. For CHANGES to a value, this value will be carried forward.
Date_Verified	Date	7	The last time we checked this information against the live system.
Date_Obsolete	Date	7	This is when this record is obsolete. This can be because the value changed, or the `Item_Name` no longer exists.

Table 2: `System_State` Oracle Table Definition

information such as a hostname or ip address do not get index pages.

Services and Servers

The key to the entire project is identifying all of the services we provide. These services range from our AFS cell and DNS, to license serving for applications, to information services such as email and usenet. These services are generally provided by one or more server machines. It is important to maintain the distinction between the "service" we are describing, and the "servers" (machines) that we are actually using to provide the service.

Each service has a number of attributes. One of the important attributes of each service is its priority. The priority will help the operations staff determine the order to try to solve problems when faced with more than one, and even if a late night page or phone call to the systems administration staff is appropriate. For example, a failure of the domain name service will result in problems all over campus, affecting hundreds, or even thousands of users, as compared to a failure of the AutoLev[6] license server which will only inconvenience a few people. Thus, we will give a

[6] I am not quite sure what AutoLev is, but according to the license server logs, only a few people use it.

Service Priorities	
Critical	Provides an essential service to the campus community. Many people are impacted by a failure. An example would be the Domain Name service.
Business High	Likely to be an administrative service, these are needed during the business day.
Academic High	These are things used in the curriculum, often in the classroom. When classes are in session, these have to be operational.
Moderate	A failure here will inconvenience a number of people, but does not warrant heroic efforts at recovery that the critical or high might require.
Low	Services that are nice, but not needed for the overall mission. These may be very lightly used applications, or more recreational type applications.
Experimental	Generally not in the repair queue at all. The person deploying the service may be interested in problem reports, but no one else.

Table 3: Service priorities.

Name	Type	Size	Description
Service_Id	Num	22	A Simon.Peoplecount.Nextval value that is used as a the primary key for the service management system. Many other tables will reference this column.
Service_Name	Char	64	The actual name of the service, such as "Domain Name Server" or "Xess License Server".
Service_Type	Char	32	An optional service type to be used in ordering reports and collecting similar function together.
Priority	Char	32	A rough indication of the priority of this service in general.
Clerk	Num	22	The Simon.People.Id of the person to create or last change this record.
Effective_Date	Date	7	The date when this record was created or last changed.
When_Inserted	Num	22	The Simon.Transcount.Nextval of when this record was inserted into the database. This makes the Effective_Date somewhat redundant, but that format is easier to display.
When_Updated	Num	22	The Simon.Transcount.Nextval of when this record was last changed. When_Marked_For_DeleteNum22T{ The Simon.Transcount.Nextval of when this record is considered obsolete. It may have been replaced, or simply considered to be deleted.
Comments	Char	2000	A place to hold a short description of this service. This information is included in some reports and on web pages and the like.

Table 4: `Service_List` Oracle Table Definition

higher priority to restoring the DNS service rather than restoring the AutoLev service. Our current list of priorities[7] is listed in Table 3. These priorities are likely to be revised once we have worked with them for a while. The priority classifications could be of interest to our users, but could potentially be a political nightmare.

Another attribute of a service that we find useful is a service type. These are just general categories like "File Server" which would include our AFS file servers, Novell file servers, NFS exports, and so on. Other categories include "Name Servers," "License Servers," "Unix Commands." These types are used for grouping sets of services together in some of the index web pages, or when searching for a specific service. This information is stored in the `Service_List` table (see Table 4).

All of the services need to be defined manually, so we needed some sort of program that our system administration staff could use. Since all of this work is being done in Oracle, we were able to use SQL*FORMS[8] to come up with a `Service List` form (see Figure 1) to enter and update information on services. The form allows our staff to define new services, as well as search for and edit existing services. When entering a new service, once the service name has been entered, the priority and service type values are selected from lists of values. This ensures some consistency and simplifies grouping for output. There is also a space available for comments on the service; although the box on the form looks small, we currently allow comments up to 2000 characters long. An editor for this field is just a keystroke away. The form automatically takes care of recording who made the latest change and when they did this.

There are many other things we want to associate with services. Unlike that attributes listed above, which occur only once per service, the others may have more than one entry per service. To help with maintaining these, the form has four buttons along the bottom that will pop up a sub window to allow you to edit the methods, documents, contact information and dependencies on other services. The data in these sub windows are linked to the current record being displayed in the `Service_List` window. This makes it simple to step through services and see the related information. The fifth button will bring up the `Servers` form, but that is a stand alone form that does not track the current record in `Service_List`. Methods and Servers will be described in later sections.

[7]Since we are college, the overall mission is to educate students and do research. With the possible exception of our phone switch, non of our operation deals with life-safety issues.

[8]SQL*FORMS is part of the Developer/2000 package, an Oracle product that allows you develop GUI programs quickly and easily. Our current release runs on both X and Wintel platforms and the next release will generate JAVA for web applications.

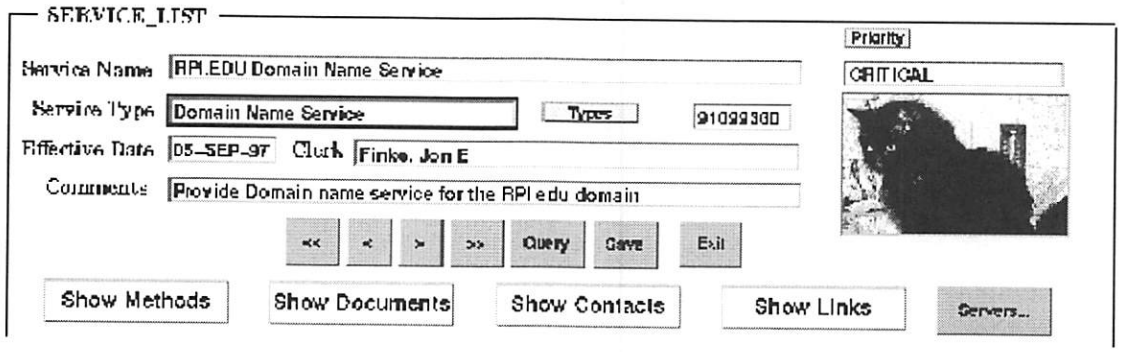

Figure 1: Service List Screen

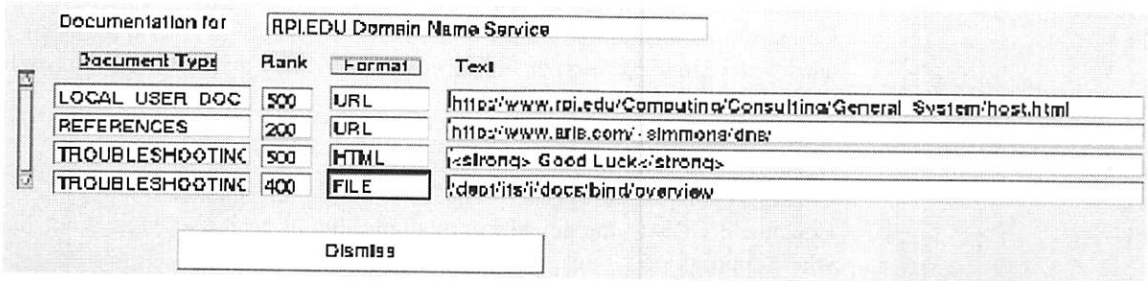

Figure 2: Service document entry/edit screen.

Service Documents

When you press the `Show Documents` button the `Service_List` form, a document entry/edit window pops up (see Figure 2). Unlike the master form which only displays a single record at a time, this form displays up to four records at a time. If there are more, you can use the scroll bars to move through them. Each document has four attributes of interest, the document type, a rank, a format and the actual text or reference.

As with service types, document types (see Table 5) are generally used for sorting and indexing on the output pages. However, it would be trivial to generate lists of services that are missing specific types of documents, such as "Troubleshooting." From a management perspective, this could go a long way to ensuring that we have at least some troubleshooting information for each of the services we offer. If we generalize our definition of "service," we can also use this to provide a general framework to hold much of our system documentation.

The next document attribute is the `Rank` field. This is a number between 1 and 999 that is used to order documents when more of one type is available for a specific service. Along with rank, we also include a document format (see Table 6). Since our primary output format for the service information is HTML, we use the document format to determine how to provide hot links when possible. This allows us to link in many of our existing documents, most of which are plain text files ("FILE"), and a few already on the web("URL").

The final field in the document screen, is the `Text` field where the actual reference or even the document text can be stored. As with the `Comments` field from the master screen, the text can be up to 2000 character in length, allowing for short in-line documents or very long URLs.

Service Contacts

One document that is usually out of date, is our list of contacts for each of our applications and services. Originally, this was a flat file that would get updated once or twice a year. An additional limit was that it only listed technical contacts for the application. In our current operation, primary support is often provided by one our two staff members, with other people providing backup support. In addition, the were often one or more folks from other departments who might provide other kinds of support. Another problem with this list, was that names were not always entered the same way, so even if you used `grep` to find your assigned areas, typos, misspellings and nicknames

Document Types	
Installation	Information on how to install the given service; generally from the vendor.
Local_Install	Local notes on installing the service here.
Local_User_Doc	Locally produced documentation that might be of interest to a user of this service.
Operational	Document for use by operations or systems administration staff; standard procedures
Overview	A document that briefly describes WHAT this service is.
References	Pointers to other relating information that may be of interest.
Troubleshooting	Information on correcting problems with this service.
Vendor_User_Doc	Documentation supplied by the vendor that might be of interest to users of the service.

Table 5: Document types.

Document Formats	
Dvi File	Document in DVI (TeX) format. (File Reference)
External	Contains a general reference to a book or other offline document.
File	Contains a pathname that points to the file, presumably in AFS space. This is used when due to location or file permissions, the doc in question is not reachable via a web server/browser.
Html	Contains HTML formatted text.
PostScript File	Document in Postscript format (File reference)
Text	Contains plain text, no HTML or other formatting information.
Url	Contains a URL to the actual information. Should be extracted as a hot link where appropriate.

Table 6: Document formats

could result in your missing some of your assignments.

To automate this, we added another database table, `Service_Contact_List`, which is used to record `Service_Id, Service_Type, Person_Id` triples. This is accessed by pressing the `Show Contacts` button on the bottom of the master form. The `Service_Id` field provides the linkage to the specific service. We have defined a few acceptable contact types (see Table 7) and the `Person_Id` links to specific person. Since we have the campus phone directory in the same database, when we map back to a person's name, we can also include their current phone number and email address. This also makes it trivial to generate a list of services assigned to each person. This can be especially helpful when someone leaves, we quickly can determine what areas need support.

Service Dependencies

We want to record the interdependence between services; for example, our PC printing service depends on the proper functioning of our Kerberos authentication service. If both are down, you have to first bring the authentication service. Interest in establishing the service dependency chart was greatly increased when we had a UPS failure and all of our servers were turned off at the same time (the first time this had happened with this configuration.), and there was concern that we might not be able to bring the whole set of systems back up.[9] Like the contact information, the service dependency window can be accessed by pressing the `Show Links` button on the bottom of the master form. Again like the contact info, the dependency information is stored in `Service_Id, Dependency_Type, Service_Id` triple. The second `Service_Id` is in fact the service id of the service that the current one depends on. One interesting offshoot of this, is that you can now have dependency chains! For example, in order for Pro/ENGINEER to run, it needs to contact the Pro/E license server. However, the Pro/E license server required that the Domain Name service be running. As a result, Pro/ENGINEER indirectly needs the DNS in order to start up. Fortunately, this dependency chain is easy to detect, so when the list of required services for Pro/ENGINEER is generated, not only are the direct dependencies (Pro/E License Server) listed, but the second order dependencies (DNS), and even higher are included automatically.

There is more than one kind of dependency though. In the previous example, Pro/ENGINEER needed the license server to start running, but once it is running, it no longer needs it. Some services, such as our SAMBA[10] service requires that the AFS service be up and running all the time. At the other end of the spectrum, we have services like our Domain Name service that require AFS if you want to make a configuration change (as the tools are stored in AFS), but normal operation and even startup are independent of AFS. To help keep track of this, we have defined some dependency types as seen in Table 8.

[9]This might seem to be rather farfetched, but we had been faced with this exact problem a number of years ago, and had to do some quick re-cabling in order to get our file servers back up after a power failure.

[10]Samba is a software package that allows wintel users to access their Unix files.

Service Contact Types	
Admin-Support	Person to contact for administrative change to service, in the case of the DNS, this would be hostmaster.
Main-User	Person(s) who have a special interest in the service and may need to be notified in case of changes or long term problems.
Primary-Support	Primary person(s) to contact in case of an outage.
Secondary-Support	Backup to the primary person(s) to contact in case of an outage.

Table 7: Contact types.

Service Dependency Types	
Operational	The service is needed for regular operation.
Periodic	The service is needed from time to time, but things can run for a while before a failure will occur.
Startup	The service is needed at startup, but not once the server is up and running.
Administration	The service is needed to make administrative or configuration changes.

Table 8: Service dependency types

A Method to this Madness

One of the objectives of this project was to automate the discovery of servers providing any given services and ideally have ways of testing that the servers are configured to provide the service and that they are actually providing that service. There are many ways to discover servers that are, or should be, providing a service. Depending on the service in question, you may be able to query some master authority such as the Internic to find your name services, or read a license file stored in the campus wide file system to find license servers. If you are on a server machine, you can look for particular configuration files, log files, or even running processes. Other potential servers can be detected by asking the clients of the service. A quick look in the `/etc/resolv.conf` should give you a list of machines that SHOULD be running name servers, or a call to `ypwhich` should tell you who should be running `nisbind`.

In addition to detecting possible servers, given a list of servers, it would be nice to be able to test those servers to see if they are in fact operational or at least configured correctly. These test methods may be very similar to the detection methods.

Attempting to write a program that can verify the configuration of all servers at a site could be an immense task. But many large tasks can be handled if you can break them up into a bunch of smaller tasks. Given that we have identified each individual service, and identified a number of different ways to try to detect or check a service, we can finally get around to writing our actual methods. Each method can a short program, a perl script, a SQL script or any other handy technique that will perform one particular test or generate a list of potential servers. These should be easy to write, and in many cases will incorporate tests developed for other systems such as *pong* [10].

Running Methods

When the time comes to run a method, you need to consider the runtime environment for the method. When you define a method (using the `Show Methods` button in the master form, you specify both the UNIX userid and the UNIX group that should be set when it runs. This helps avoid running things as root that don't need to be root. We also make a couple of other assumptions about the run time environment, at least for the global cases; that it is running on the Simon Database machine which allow access to the database without ID/Passwords pairs and that adequate AFS and DCE credentials exist to access files of interest in the AFS and DFS cells. In addition to the runtime environment, we also need to know how to make the method report back to system. Rather than requiring that each method invoke some program to report results, we instead defined several method result types (see Table 9). Depending on the method, it can simply set the return code before exiting, or output a list of values.

Global Detect Methods

A global detect method can detect a potential server from just about anywhere. In practice, this really means that it can look in the campus file system (and has appropriate file access to do so), or can query the Simon Database to extract configuration information, or query some master server.

An example of detecting a server via the file system, would be the Matlab license server. When a Matlab client starts up, it consults a license file to find the names of the license servers. In order to extract this list, just issue the command in Listing 1. This returns a list of all the license servers that Matlab clients will try to use. In this case, we didn't have to write any programs at all, just use existing system commands.

We can also write methods that contact other information servers for the list of servers. These can be trivial, such as the first of the following examples which determines our YP servers, or the second example which is slightly more complicated which we use to find our current web servers.

```
ypcat ypservers
nslookup www.rpi.edu | \
    grep 'Addresses:' | \
    cut -d: -f2
```

The second example could in fact be more complex.[11] Ideally, we would poll each of our name servers (assuming we can figure out where they are!) and ask each of them which machines they think provide our web service. But generally, all of our name servers are pretty good about giving out the same information.

All three of the above example return a list of servers providing a given service. The first two give one server per line, and the last gives a comma

[11]This example actually has at least one fatal flaw – if there is only one address, the column is labeled "Address:," but the address of the server is ALSO printed with an "Address:" label.

Method Result Types	
Client_List	This method returns a list of server/client pairs, either hostnames or IP addresses.
Exit 0 Ok	This method reports success by exiting with a return code of 0.
Server_List	This method returns a list of server names and/or IP addresses.

Table 9: Method result types.

delimited list of servers. The `find_server_state` program can handle both types of lists.

Server Config Detection Methods

Another way to check for a particular service on a machine is to the system configuration files on that machine. For example, running a PH server[12] would require the line "%define PhServer" in the `/etc/package.config` file. The next package run would install all the parts for that server. Alternately, we could attempt to install one manually by making an entry in `/etc/inetd.conf` for the server and a cron entry to regenerate the local PH database. In any event, seeing any of those three would indicate that at least an attempt was made it install a PH server. To detect that, we could execute the commands in Listing 2. If any one of those three indicators is found, we flag the host as a potential PH server (note: the three grep commands should all be one line, joined by a logical "or").

Server Activity Detection Methods

A different approach to finding a service on a machine, is to look for indications that the server process is running. This could be as direct as checking `ps` for a process of the appropriate name, or by looking for traces left behind by a server such as log files, or messages written to system log files such as `/var/adm/messages`. You would want to be careful in checking log files that you are looking at recent log files. A quick check for an Oracle server using the `ps` command could be something like:

```
% ps aux | grep -v grep \
     | grep -q ora_smon
```

[12]"ph" is phone directory server originally developed at University of Illinois at Urbana

Client Server Detection Methods

Perhaps one of the best ways to discover what servers your machines are expecting to find, is to ask the individual machines; after all, they are trying to use the services. These checks need to be done on the client machine, either directly by accessing the client file system (reading `/etc/resolv.conf`) or running daemons (`ypwhich`) or indirectly by reading saved data from `MyState` or things like that. In this case, it is important to also record which client detected a given server, as the error may be with the client. Two approaches to finding yp servers, the first runs on a client and just returns the server name, and the second runs globally and returns client/server pairs. One danger in reading saved data, is the data may be out of date. See Listing 3.

Global Server Testing

Instead of detecting servers, we can also use methods to test if a server is currently providing a service. In general, to test a service from a central point, you need to contact it, or "ping it." There has been a lot of work in the area at other sites, and we hope to start taking advantage of that soon. At this point, we have not done a lot with testing services.

Server Config Verification

Another useful check of a service, is to see if it configured correctly on the actual server. This is similar to the configuration detection methods, except you want to verify that ALL parts are in place, and not just some of them. However, in our current installation, most services are installed with package, which attempts to install all the required files and configuration changes. Any failures in this process are reported and generally corrected pretty quickly. In general, we find that services are installed and operating, or not installed at all; configuration errors are rare.

```
awk '/SERVER/ {print }'
/campus/mathworks/matlab/5.0/distrib/etc/license.dat
```

Listing 1: Extracting license list.

```
grep -q '%define PhServer' /etc/package.config || \
grep -q 'csnet-ns' /etc/inetd.conf || \
grep -q CreatePHDir /var/spool/cron/crontabs/root
```

Listing 2: Checking installation attempts

```
ypwhich
cd /dept/its/config/admin/etc/ShippedBackFiles/etc/MyState ;
grep ypmaster *.rpi.edu | cut -d: -f1,3
```

Listing 3: Finding servers.

Local Server Activity Verification

Testing services by checking for running processes is also possible, although is often a crude check in cases where a server is hung up, but still appearing in `ps`. Most of our services fall into two categories, "reliable," where they run forever, or "flaky" where we have generally put something in place to frequently check them and restart them if needed. In these cases, automatic restoration of the service seems more important than reporting the failure.

Servers

Once a service has been defined, we can assign one or more servers to that service. This relationship can be established manually by a staff member, or can be detected automatically using one or more of the methods defined in the previous section. When a server/service relationship is set up, it is assigned a status value. For entries made by staff members, this could be something like "PRODUCTION," "TEST" or "OBSOLETE." For entries discovered by the system, it would get the value "UNVERIFIED." This is actually considered an error condition in most cases, and must be changed by a staff member. Until then, it will be included in a daily error report sent to the admin staff. The objective here is to ensure that some human has considered WHY we have a new server, to help avoid dependencies on servers we do not know about.

Since any given service may be provided by more than one server, we need to be able to store multiple server records for each service. To this end, we have defined the table `Server_List`. This table records the relationship between a service and a particular server machine. This includes the status as described above, the date when the status was changed, and who changed it, along with their comments. It also records an optional review date with comments on that. It also will record automatic detection information from the methods described above. For each method (global, server and client detect), it records the date first detected, the date cleared (if the last check failed), and the date the last check was done. In addition, for client detection, the `System_Id` of the client is also recorded. Our original intention was to have all the user interface for this

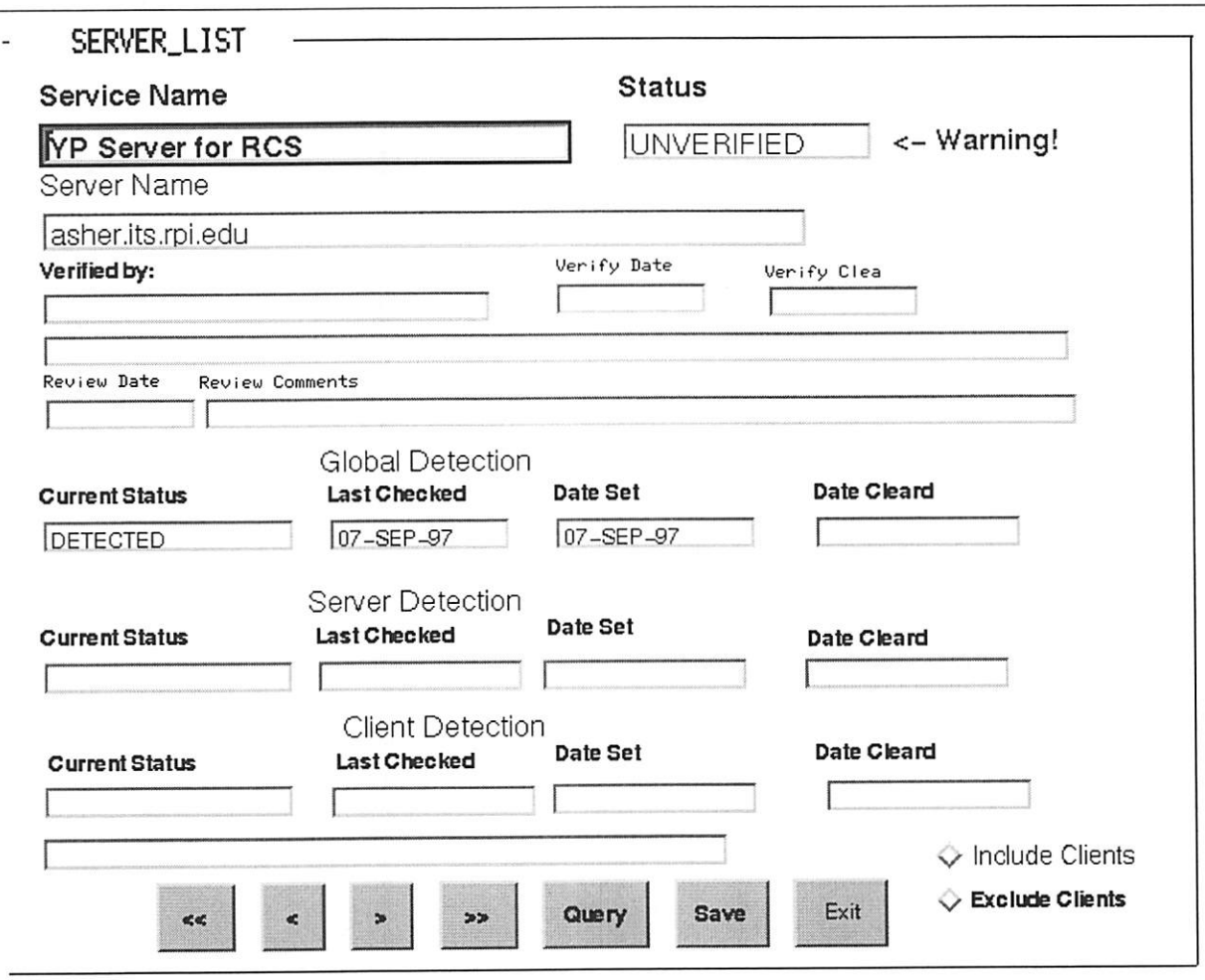

Figure 3: Server Entry/Edit Screen

project be using a web server, but some problems with getting that installed in a timely manner, resulted in us developing a SQL*FORM application, `Server_List`, to allow our staff to update the status values for each server record (see Figure 3). The upper half of the form has fields that can be updated by staff members, and the lower half has the three sets of detection results. One of the side effects of the client detect process, is that we get a record for each client of a particular server/service pair. Rather than display all of these records on this form, there is a switch at the bottom of the form that allows the user to include or exclude the client detect records. However, since client detection information may be important, when we record client detection information (date set, etc), we also record the status and date checked and date set in the "master" record (the record that has the global and server detect information, as well as the manual information. We don't expect our staff to go in and validate each client record individually. In order to help with the validation process (where a staff member changes the status from "Unverified" to something else), we bring up a warning message and arrow, alerting the user to the questionable status.

Weaving Everything Together

With the information described above, we are able to generate the server lists requested by the NOC staff. Not only the list of machines providing each service (which is handy when the call comes in that says the XXX service is broken), we can easily generate the list of services provided by any given machine (which is handy when we learn that a particular machine has gone down). These lists are generated automatically and sent to the NOC as needed. Although hardcopy lists may seem old fashioned, they are nice to have when you are reading them by flashlight, attempting to determine what order to power up things once the UPS is repaired.

Server (System) Pages

We wanted more than that though. By generating HTML pages, we are able to include a lot more information, not only for services, but for the servers themselves. We are already collecting all of the `/etc/MyState` information for all machines, so this provides the basis for servers pages, in fact, all we need to do is add the services information as part of the MyState page generation and we have a pretty detailed picture of any given machine. Figure 4 is partial view of one of the server (actually, system) pages. This includes some system information (machine type, location, serial number) drawn from the Hostmaster database, the server information, listing two services that this machine provides, and continues on to DNS information, and the MyState information, allowing our system administrators to get a pretty complete

netserv1.its.rpi.edu

Comments on the format and layout of these pages, as well as ideas for improvements, should be directed to finkej@rpi.edu Page generated at:Sun Sep 7 16:02:50 1997

System Info

- Location: VCC Machine Room
- Hardware: Sun SPARC4
- SN: 605F00E7

Server Info

Priority: CRITICAL

- RPI.EDU Domain Name Service Status: PRODUCTION

Priority: HIGH

- Print Server Status: PRODUCTION

DNS info for netserv1.its.rpi.edu

IP Address	128.113.1.5

mystate

Figure 4: Server (System) HTML page for Netserv1

picture of the machine. In order to aid navigation, we include a lot of links. We have both generic links back to the appropriate index pages (if you select "Critical," you will go to the index page of critical priority services.) and specific links such as "RPI.EDU Domain Name Server," which takes you to that page (see Figure 5).

Service Pages

This page contains a lot of information about this service. As you can see in Figure 5, it starts with some general information about the service, a short description, followed by the priority, service type and contact information. The contact information automatically includes the person's phone number and email address if available. The email address is even set up as a `mailto:` URL. The next section lists which machines are providing that service, along with the status, and who verified that status and when. Clicking on the machine name will bring you to the page for that machine (like Figure 4). The next section lists the services that THIS service requires, and what type of dependency, along with any comments on that dependency. Following that, we include whatever documentation we have available for that service, ordered by type; again, we have provided hypertext links where possible. Finally, we have a list of services that rely on this service. As with the server(system) pages, we try to provide all the relevant information about a service in one neat package, and provide quick navigation to the related topics.

Other Pages

We also generate a number of other types of pages. We can generate document pages, sorted both by document type (troubleshooting, operation, etc),

General Information

Provide Domain name service for the RPI edu domain

- Priority: CRITICAL.
- Classification: Domain Name Service
- Primary Contacts: David K Hudson hudsod@rpi.edu 8836
- Secondary Contacts: Jon Finke jfinke@rpi.edu 8185

Servers providing this service

netserv1.its.rpi.edu
- Current Status is: PRODUCTION
- Verified by Jon Finke jfinke@rpi.edu 8185 on 07-SEP-97

netserv2.its.rpi.edu. vccprntserv.its.rpi.edu
- Current Status is: PRODUCTION
- Verified by Jon Finke jfinke@rpi.edu 8185 on 07-SEP-97

Services required by this service.

- Direct dependence servers.
  Simon - ADMINISTER
  Hostmaster Data is stored in Simon.

Documentation

REFERENCES
http://www.aris.com/~simmons/dns/
TROUBLESHOOTING
file:/dept/its/i/docs/bind/overview
LOCAL_USER_DOC
http://www.rpi.edu/Computing/Consulting/General_System/host.html

Services that rely on this service.

- Direct dependence on this service.
  Sun Compiler License Server - OPERATIONAL
  Needs to look up host names
  Simon SqlServer - OPERATIONAL

Figure 5: Service HTML page for RPI.EDU Domain Name Service

and by service type (Unix Command, License Server, etc). We also generate Contact pages, so you can easily check what services are assigned to a given individual. Using the relational database to store the information gives us many options in organizing and displaying the data; hopefully we can be all things to all people.

Future Directions

Our immediate challenge is to populate the database with the rest of our services. I expect we will expand the definition of "service" to include all of the applications we support, if for no other reason, than to simplifying tracking the contact information and documentation for each application.

We will also be writing programs and SQL scripts to provide cross checking for errors and inconsistencies. These will range from missing information such as contact person and troubleshooting documentation, to differences found by different types of detection. If a server is detected with one method and not with another, there is a sure sign that something needs to be investigated.

Once we get the latest version of the Oracle Web server installed, I expect to make the forms available as web pages. We will also be investigating generating at least some of the pages in real time.

We are also considering dumping the entire service/server web tree to a CD which would be available for use on a stand alone machine (handy for when you have major problems), or even for the on call person to take home to run on their home machine. While this has the drawback of the CDs getting out of date, they still may be handy as a reference, especially when network access is slow or missing altogether.

References and Availability

All source code for the Simon system is available for anonymous FTP. See ftp://ftp.rpi.edu/pub/its-release/simon/README.simon for details. In addition, all of the Oracle table definitions are available at http://www.rpi.edu/campus/rpi/simon/misc/SIM2/SIMON-index.html .

Given that much of the output of this system is web based, it would be very nice to be able to make all the pages generally available. If nothing else, it would make it a lot easier for our own staff to access it. However, before we can release the pages, we need to assure ourselves that releasing it will not result in the unplanned release of confidential information or reducing site security. If there is sufficient demand, I can release a subset of the pages for demonstration purposes with sensitive material removed.

Author Information

Jon Finke graduated from Rensselaer in 1983, where he had provided microcomputer support and communications programming, with a BS-ECSE. He continued as a full time staff member in the computer center. From PC communications, he moved into mainframe communications and networking, and then on to Unix support, including a stint in the Nysernet Network Information Center. A charter member of the Workstation Support Group he took over printing development and support and later inherited the Simon project, which has been his primary focus for the past six years. He is currently a Senior Systems Programmer in the Server Support Services department at Rensselaer, where he continues integrating Simon with the rest of the Institute information systems, and also deals with information security concerns.

Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via http://www.rpi.edu/~finkej.

Bibliography

[1] Paul Anderson, "Towards a high-level machine configuration system," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pages 19-23, USENIX, San Diego, CA. September, 1994.

[2] Jon Finke, "Automated userid management," *Proceedings of Community Workshop '92*, Papers 3-5, Rensselear Polytechnic Institute, Troy, NY, June, 1992.

[3] Jon Finke, "Relational database + automated sysadmin = simon,' Invited Talk for SUG-East 93, Sun Users Group, Boston, MA, July, 1993.

[4] Jon Finke, "Automating printing configuration," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pp 175-184. USENIX, San Diego, CA, September, 1994.

[5] Jon Finke, "Sql_2_html: Automatic generation of html database schemas," *Ninth Systems Administration Conference (LISA '95)*, pp 133-138. USENIX, Monterey, CA, September, 1995.

[6] Jon Finke, "Institute white pages as a system administration problem," *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pp 233-240.USENIX, Chicago, IL, October, 1996.

[7] Jon Finke, "Monitoring application use with license server logs" *The Eleventh Systems Administration Conference (LISA 97) Proceedings*, USENIX, San Diego, CA, October, 1997.

[8] Stephen E. Hansen and E. Todd Atkins, "Automated system monitoring and notification with swatch," *USENIX Systems Administration (LISA VII) Conference Proceedings*, pp 145-156., USENIX, Monterey, CA, November, 1993.

[9] Dr. Magnus Harlander, "Central system administration in a heterogeneous Unix environment: Genuadmin," *USENIX Systems Administration*

(LISA VIII) Conference Proceedings, pp 1-8, USENIX, San Diego, CA, September, 1994.

[10] Helen E. Harrison, Mike C. Mitchell, and Michael E Shaddock, "Pong: A flexible network services monitoring system," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pp 167-171, USENIX, San Diego, CA, September, 1994.

[11] Todd Miller, Christopher Stirlen, and Evi Nemeth, "satool – a system administrator's cockpit, an implementation," *USENIX Systems Administration (LISA VII) Conference Proceedings*, pp 119-130, USENIX, Monterey, CA, November, 1993.

[12] John P. Rouillard and Richard B Martin, "Config: A mechanism for installing and tracking system configurations," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pp 9-18, USENIX, San Diego, CA, September, 1994.

[13] J Schonwalder and H Langendorfer, "How to keep track of your network configuration," *USENIX Systems Administration (LISA VII) Conference Proceedings*, pp 189-193, USENIX, Monterey, CA, November, 1993.

[14] Rex Walters, "Tracking hardware configurations in a heterogeneous network with syslogd," *Ninth Systems Administration Conference (LISA '95*, pp 241-246, USENIX, Monterey, CA, September, 1995.

[15] David C. M. Wood, Sean S. Coleman, and Michael F. Schwartz, "Fremont: A system for discovering network characteristics and problems," *USENIX Technical Conference Proceedings*, pp 335-347, USENIX, San Diego, CA, January, 1993.

Chaos Out of Order: A Simple, Scalable File Distribution Facility For 'Intentionally Heterogeneous' Networks

Alva L. Couch – Tufts University

ABSTRACT

In large networks, heterogeneity in hardware, operating systems, user needs, and administrative responsibility often forms boundaries that inhibit sharing of information, expertise, and responsibility. These boundaries can divide networks into 'feudal fiefdoms' of administrators, each with a disjoint domain of responsibility. DISTR is an easy-to-use file distribution tool for homogeneous networks that also provides controlled file transfer between disparate architectures and administrative domains. Using DISTR, administrators of unrelated networks can collaborate to reduce duplication of effort while retaining control of their own networks. DISTR's controls allow collaboration, cooperation, and camaraderie to evolve, not from grand and imposed designs, but from informal and serendipitous commonalities of mission and purpose.

Introduction

Computers, like people, have to share a common language in order to communicate. To share information, they have to agree on the meanings of the files they share. Unfortunately computers, like people, are often prohibited from communicating by what are almost 'ethnic' boundaries: diverse hardware, diverse operating systems, or even diverse user and administrator needs.

While we are all familiar with heterogeneous networks of machines with diverse hardware or operating systems, there are also networks with homogeneous hardware and operating systems but diverse configurations or administrative responsibilities. Users often need many different configurations of the same hardware and operating system. An often overlooked form of heterogeneity is that induced by lines of authority, control, and responsibility. Mimicing the structure of a large organization, a large network of relatively identical machines is commonly organized as several 'feudal' subnetworks, or 'fiefdoms,' each with a different primary administrator or administrative group.

Distributing shared information is relatively easy when one does not have to cross boundaries induced by disparities in hardware, software, user needs, or administrative responsibility. There are plenty of solid approaches to distributing databases, files, and software. However, whenever one needs to cross even one of these boundaries, many problems arise. Different operating environments require different formats for the same information, but formatting information correctly is the least of our problems. It is much more difficult to deal with user requirements and administrative boundaries. Current tools for file distribution cannot be configured at the level of detail necessary to describe and respect these boundaries.

In this paper, I describe a new tool for file distribution, DISTR, specifically designed to deal with heterogeneity in hardware, software, user needs, and administrative boundaries. Written in Perl-5 [21] for portability, DISTR has a simple syntax that eases simple file distribution tasks in homogeneous networks. For advanced users, the functions of DISTR can also be extended by writing custom Perl scripts. These scripts can authenticate data transfers, keep records, distribute files to large networks quickly, and transform data appropriately both before and after each transfer through a heterogeneity boundary.

Distribution Systems

There are currently many ways to distribute the contents of files on a network. These include generic tools that will distribute any kind of file, and domain-specific tools that distribute only particular kinds of files or work only within particular kinds of networks.

The simplest kind of file distribution is to use a command that copies a file from one place to another. Remote copy commands such as the UNIX remote copy command `rcp` and a recent improved version `rsync` require the user to have an account on the target machine and the rights to modify the file to be copied. To update system configuration files, the user must have `root` access to the target machine. Secure remote command facilities including SSH [23] better insure the identity of the updater but suffer from the same need to grant `root` access. Alas, this simple approach works for very small networks but is not scalable to large networks or applicable to non-UNIX systems.

General-purpose UNIX file distribution schemes such as RDIST [7] automate the process of using

remote commands on a server to copy files to remote clients. RDIST can both update files and execute remote commands necessary before the new versions take effect. Reverse file distribution systems like RevRdist [25] and PC-Rdist [17] perform the same function but reverse the roles of client and server. These execute on each client at boot time or some other convenient time and copy files from a server to the client, accessing the server's files using a file sharing mechanism such as AppleShare or the Network File System(NFS) [20]. Before updating a client file, all of these tools check that it needs updating by comparing its size, creation time, and other attributes with those of the master copy on the server. The goal of these tools is to allow one administrator to exclusively control more machines than is usually possible.

Software package distribution utilities such as Depot [6, 15], opt_depot [1], Depot-Lite [19], Cicero [4], and their variants accomplish the same task as RDIST, but limit their problem domains to the special problem of distributing usable software. This limitation allows these tools to perform more complex tasks than generic tools like RDIST while requiring less effort to configure. Variant file distribution schemes such as Local Disk Depot [22] add the ability to customize each client machine's software environment, to avoid installing unused software and conserve disk space.

Configuration trackers [3] document the contents of specific UNIX configuration files on clients in a heterogeneous UNIX environment. A database on a server describes the similarities and differences between clients. A configuration tool uses remote commands to assure that clients' configurations agree with the database. This again is what RDIST does, limited to a very specific problem domain to simplify usage and avoid distribution errors.

Domain-specific UNIX database services such as NIS [20] and NIS+ [18] automatically provide data from tabular databases on a network. NIS client machines must choose one server to provide database information. NIS+ adds the ability to query multiple servers and administrate large databases in distributed, manageable pieces. As both NIS and NIS+ only work on databases in which each line of a table has a unique key, neither is suitable for providing files without a tabular structure.

Portable domain-specific database services such as the Lightweight Directory Access Protocol(LDAP) [24] and Domain Name Service(DNS) [2] overcome many of the portability limits of NIS and NIS+. These are designed to provide information to all hosts in a heterogeneous environment but are too specialized and optimized for their problem domains (mail addresses and internet names) to be useful for general-purpose distribution of any other information.

A pattern develops in examining all these approaches. All except NIS+ take their information from one master server, even if slave servers replicate its contents. Each approach is either good for distributing a specific kind of information to different kinds of client machines, or all kinds of information to roughly the same kinds of client machines. None of these tools addresses the general problem of distributing the contents of arbitrary files in heterogeneous domains containing a mix of UNIX and other operating systems. There is a good reason for this; the general file distribution problem is very difficult!

Developing trust

"It would be nice" if we could all trust each other and cooperate on what is obviously a common goal. Unfortunately, trust is difficult to encourage among diverse groups. It is especially difficult to encourage when trusting another group to provide services means giving all of them the ability to change anything at all on any machine in your own network! But this is precisely what most current file distribution tools require. Trust is much easier to develop incrementally from small, serendipitous commonalities of goals and purposes. With proper tools, trust can grow from 'grass roots' upward, from good experiences in allowing cooperative people the access they need to make a difference.

For example, many fiefdoms are small enough that they do not run their own name service. This is instead handled by a central service group that determines, e.g., the contents of the name service configuration file `resolv.conf` for the feudal network. But when this file changes, the central group may not have privileges to install it within the fiefdom. Typically someone from the central group mails each new version to someone inside each fiefdom and tells the internal person to install it.

This situation wastes the time of both the central group and the fiefdom administrator. Central services should be able to change `resolv.conf` without changing anything else. The fiefdom administrators should be able to trust central services to update this file, regardless of whatever else they think about central services.

Desires

After a several year struggle using existing file distribution tools and writing front-end tools that extend their capabilities, I set out to implement a new file distribution tool DISTR that does not suffer from their limits.

My priorities were quite simple at the outset. I wanted a file distribution mechanism that allows client machines complete control over which files they accept. I have had many problems with particular users who need customizations outside the limits of my configuration management scheme. In a hurry, I wanted something that will allow me to turn distribution actions on and off at the client end and modify them according to personal taste.

I wanted a file distribution mechanism that is symmetric in the sense of supporting both master to slave and slave to master requests. I like the master to slave administration model, but I have historically had much trouble keeping all hosts synchronized when lone hosts suffer hardware failures and miss receiving files.

I also wanted a mechanism that utilizes portable file names rather than names particular to one operating system. I chose the dotted name notation of many tools as a starting point. Each portable name is a point of sharing between two hosts; regardless of whatever else they can agree upon, they can agree upon a name for the file being exchanged.

Finally, I wanted a tool that is very easy to use at the outset and hides advanced features from novices. I did not want to have to think very much about routine distribution tasks. But I also wanted to be able to perform arbitrarily complex tasks using the tool with more effort.

Beginning from these needs, I crafted DISTR. What evolved is a much more complex entity than these needs prescribed.

Using DISTR

DISTR's basic usage is simple if a bit cumbersome. In order to use DISTR, one has to install DISTR's daemon `distrd` and DISTR's user front end `distr` on each host to be involved in providing or receiving files. The daemon runs at all times awaiting requests. A host that provides a file is called a *server* while a host that receives one is called a *client*. In DISTR, there is no real distinction; any host can be a client of some files, a server of others, and a server and client of still others.

Next, one creates a configuration file `distr.conf` on each host describing the files that this host is allowed to receive or provide. On each server, `distr.conf` contains lines like:

```
aliases.export.source =
        '/usr/lib/aliases';
aliases.export.clients =
        ['mine', 'yours'];
```

These particular lines give DISTR permission to export the local file `/usr/lib/aliases` to two clients `mine` and `yours`.

The configuration file of each client receiving this file must have a matching set of lines like:

```
aliases.import.target =
        '/etc/mail/aliases';
aliases.import.servers =
        ['theirs'];
```

These lines give the client permission to accept a file from the server `theirs` and put it into the local file `/etc/mail/aliases`.

With configuration files in place, the user then requests distribution actions by executing commands like

```
# distr aliases.import
```

on a client to import the alias file from a server or

```
# distr aliases.export
```

on a server to export the alias file to all clients.

Using DISTR is very different from using RDIST. Both client and server must have configuration files, and these must agree on what gets exported from one machine and imported into another. For a file transfer to be able to occur, server and client have to agree on a *portable name* (such as `aliases`) both will use to refer to the file, and each must define enough *parameters* for the transfer to allow it to occur. Parameters have names that are prefixed with the portable name of the file.

At bare minimum, server and client have to specify two parameters. On a server, `export.source` is the name of a file to provide and `export.clients` is the familiar RDIST-like list of all clients to which to provide it. On a client, `export.target` is the name of a file to accept and `import.servers` is a list of all servers from which the client can potentially obtain the file. One denies a host the privilege to install a file by simply omitting the host from the list of authorized servers for a file or, even more simply, omitting the file name from all DISTR declarations for the client.

Once this information is specified, one can initiate a file transfer from either host. Like RDIST, one can tell the server to transfer a file to clients. Like RevRDist, one can tell a client to fetch a file from a server. Unlike RevRDist, DISTR will search for a valid copy of the file sequentially on all servers on its list, stopping when it finds and manages to fetch one.

Simplifying Syntax

DISTR provides several syntactic features to make configuration file syntax less cumbersome. Complex specifications are easier to type by using a name scoping notation. The above server configuration can also be written as

```
aliases.export {
  source = '/usr/lib/aliases';
  clients = ['mine', 'yours'];
}
```

Inside the braces, names are prepended with the prefix given before the braces. Scopes nest to arbitrary depth, so one could also write the server configuration as

```
aliases { export {
  source = '/usr/lib/aliases';
  clients = ['mine', 'yours'];
}}
```

Using Inheritance

DISTR's parameter values may be defined or inherited. A defined parameter has a specific value listed in DISTR's configuration file. An inherited parameter gets its value from another defined parameter. A name with no defined value inherits the value of its *longest defined suffix*. If `c.d` is defined and `b.c.d` and `a.b.c.d` are not, then `a.b.c.d` inherits the value of `c.d`. This allows the user to assign default values to parameters rather than typing explicit values for all distribution actions.

For example, we can write:

```
export.clients =
    ['mine', 'yours'];
aliases.export.source =
    '/usr/lib/aliases';
group.export.source =
    '/etc/group';
```

to implicitly export *all* source files to both clients. Attributes `aliases.export.clients`, `group.export.clients`, and every other attribute whose name ends in `export.clients` inherit their values from the 'global' definition for `export.clients`. Thus these attributes do not need to be specified explicitly.

In turn, the rest of the configuration file need not mention `export.clients` again and can consist only of mappings between portable names and names of files to export. In homogeneous environments with one server and clients that all receive the same files, DISTR configuration files are thus much shorter in length than comparable configuration files for RDIST.

This notion of inheritance is almost exactly *backward* from notions of inheritance in object-oriented languages such as Java [11] and JavaScript [12] that use the same kinds of namespaces. Object-oriented inheritance helps one specify rules that define classes of objects but does not create any instances of those classes. Our inheritance functions instead help us specify *instances* of classes that never get explicitly defined as classes! Rather than defining a set of classes each of which can have an unlimited number of instances, DISTR's syntax defines an unlimited number of instances, leaving the classes implicit in that definition!

DISTR's Syntax

DISTR's configuration file defines values for selected attributes. Each attribute is represented by a dotted name and can represent either an action or parameter. The value of an attribute is the value of a Perl expression. This value can be any kind of Perl scalar, including a reference to an array, to an associative array, or to a function. An attribute whose value is a reference to a Perl function is called an *action*, while an attribute with a non-function value is called a *parameter*. Internally the only distinction between actions and parameters is that when other actions request their values, actions are implicitly invoked as functions while parameter values are simply returned to the requester. The set of all defined attribute names and their values is called DISTR's *namespace*.

DISTR distributes files solely by invoking actions defined in its namespace. These actions query the namespace to determine their own operating parameters. As inheritance works for actions as well as parameters, an action can be invoked by many different names with differing prefixes. The prefix on the name by which an action is invoked determines the parameters it fetches from the namespace.

DISTR provides two default actions `export` and `import`. These are never invoked directly, but always through inheritance. The names by which they are invoked determine which files are exported or imported. For example, invoking `export` as `aliases.export` (through inheritance) causes it to use the parameters `aliases.export.source` and `aliases.export.clients`. The values of these may themselves be inherited. If `export` is called by any other name its parameter names change to match.

Customizing DISTR

In configuring DISTR, the user can override the definitions of *any* action, including default ones such as `export` and `import`. However, these are very complex actions. For proper function of DISTR, these actions have to have roughly the same form regardless of customizations. For this reason, high-level operations such as `import` and `export` are phrased in terms of lower-level component operations. These lower-level operations can be individually overridden to alter specific parts of DISTR's behavior. The `import` and `export` routines are actually *algorithmic skeletons* [5] that hide the complexities of DISTR's functions while allowing one to customize anything easily.

For example, the default `import` action looks like this:

```
import = sub {
 if (&some('import.needed')) {
  if (&some('import.authentic')) {
   if (&some('import.before')) {
    if (&some('import.method')) {
     &some('import.afterSuccess');
    } else {
     &some('import.afterFailure');
    }
   } else {
    &some('import.afterBeforeFailure');
   }
  } else {
   &some('import.afterDenial');
  }
 }
};
```

The DISTR library function `some` looks for a parameter value relative to the name under which the current action was invoked. If `some` finds a value representing an action, it executes that action and returns the result of execution. If `some` finds a non-action value, it simply returns that value. The result is that the above code executes several actions in order:

- `import.needed` determines whether an import is needed at this time.
- `import.authentic` determines whether importing should be allowed.
- `import.before` prepares for importing, including backing up old versions, etc.
- `import.method` actually imports the file.
- `import.afterSuccess` does import post-processing if the import succeeds, such as updating other related databases.
- `import.afterFailure` performs cleanup after import errors, including restoring old versions of files.
- `import.afterBeforeFailure` performs cleanup after fatal pre-import errors.
- `import.afterDenial` takes action concerning security failures, including notifying administrators, etc.

By default, only `import.needed`, `import.authentic`, and `import.method` are defined as actions. The rest have value 1 (to disable them) and are provided explicitly for the purpose of customizing the import process.

The typical administrator, concerned with only a few of these phases, can choose to override actions in chosen phases and retain defaults for every other phase of importing. For example, to run `newaliases` after importing a mail aliases file, one would write

```
sendmail.aliases.import.afterSuccess =
  sub {
   system('/usr/ucb/newaliases');
  };
```

or its equivalent for your own flavor of UNIX. One of the nicest things about DISTR's configuration model is that the administrator of a local machine who does not control alias file distribution can add local modifications before compiling the file, e.g., concatenating the alias file with a local one before making it take effect.

PGP Authentication

DISTR's default authentication is host based like RDIST's. Clients maintain lists of hostnames and addresses from which requests will be accepted, though these lists are private to DISTR and not used for other purposes. DISTR also allows authentication of the creator of a distributed file as a person, using PGP signatures to determine the identity of the creator.

To authenticate every incoming file via PGP 2.6.2, one can write

```
import.authentic = \PGPauthentic;
```

to use the builtin PGP authentication function for every import transaction. One must also define two PGP parameters: list `signers` of PGP names of people allowed to provide files for distribution and the name of a keyring file DISTR can use to validate file signatures. These parameters are inherited and can change for different imported files.

Files to be authenticated must be PGP signed on the source machine using detached signatures. A file's detached signature, if available, is forwarded as part of any export request or import response. If PGP authentication is in effect, a file will only be imported if its signer matches one of a list of PGP users authorized to import files.

DISTR uses PGP for authentication only. Files are not encoded or encrypted using PGP, but remain in plaintext. There are two good reasons for this apparent oversight. Detached signatures are easy to use and do not prohibit a host receiving them from ignoring them and falling back to host-based authentication if the host does not have PGP. Encrypting each file using PGP would also require that the receiving daemon have access to a *private* key for purposes of decrypting it. Since there will not be a user available to type a passphrase, that private key would have to be stored on the host somewhere and vulnerable to discovery. In my view this use of PGP encryption provides more of an illusion of security than actual security. DISTR transmits all its files in plaintext and is not suitable for transferring sensitive information, and no use of PGP will solve this problem satisfactorily.

While this approach solves the problem of authenticity, there is still a serious security problem in using this very simple form of PGP authentication. Any file, once signed, can be successfully included in any request the signer is permitted to make. This means that a devious person with possession of a signed file can use replay attacks to corrupt every file the signer is permitted to change!

There is only one stateless PGP solution to this problem that is under development. The signer can also provide a 'certificate of intent' listing where the file should be installed. If this is also PGP signed, a host receiving the file and certificate can insure that it is utilizing the file in the way the creator intended.

Even this is insecure, because a devious person with access to an older version of a file, its certificate, and their signatures can use them to initiate a rollback attack that resets a target file to an old configuration. This can re-open security holes or deny services based on improper machine configurations.

There is again only one (almost) stateless PGP solution to this problem that is under development. DISTR can ask the initiator of a request to sign a file describing the request itself. If this file contains a time stamp and time limit on the request, and if receiving

hosts check that the request has been sent between those limits, rollback attacks can be prevented.

There are too many 'if's in this discussion. My conclusion from this is that PGP provides just a bit better security than the default host-based security, and in general is not particularly suitable for providing security in this context. Of course, with DISTR's modular structure any new available form of security can be implemented relatively quickly. One should be!

Remote execution

DISTR also can provide general-purpose remote execution capabilities. As `distrd` runs as `root`, any actions specified are privileged unless otherwise noted. This means that I can easily configure DISTR to import and execute a Perl script in a somewhat secure manner, by writing:

```
penguin.import {
 target = "/tmp/penguin$$";
 needed = 1;
 signers =
   ['Alva Couch <couch@tufts.edu>'];
 authentic = \PGPauthentic;
 afterSuccess = sub {
  my $name = &some('import.target');
  system("/usr/bin/perl $name");
  unlink $name;
 };
 afterFailure = sub {
  my $name = &some('import.target');
  unlink $name;
 };
}
```

The result of this simple hack is to force importing of any file with the portable name `penguin`, authenticate it as mine, and if authentic, execute it as a Perl script. This is not as secure as the real PENGUIN [13] remote execution utility for Perl scripts, which executes its scripts in a controlled, limited environment based upon privilege specifications. DISTR's scripts are executed with no limitations.

I do not recommend doing this. As above, this mechanism is subject to replay attacks. Once a file is signed, anyone who can gain ownership or capture it on the network can forward it to the daemon for execution.

Handling Heterogeneity

Effective communication between servers of diverse hardware and software architectures requires careful configuration of DISTR. *Any attempt* to move information from one architecture to another requires that one:

1. Understand the forms that information will take on either side of the architectural boundary, including files and their formats.
2. Design a 'portable format' for the information that can be transformed into the forms needed on either side. This may consist of information from several different files on each side of the boundary, appropriately combined.
3. Develop filters that create the portable format from files on the server side, and that transform the portable format into desired files on the client side. These filters form the bulk of DISTR's `export.before` and `import.afterSuccess` methods.

If one is lucky, the native form of information on one side of the boundary can be used as the portable form on the other. For example, suppose we are sending a UNIX database to an NT workstation. If there are many NT stations and few UNIX servers, it makes sense for the database to get translated once on the UNIX server, then propagated to all the NT stations in native form.

Scaling DISTR

DISTR's distribution algorithm can also be adapted to update large networks in minimal time by a relatively simple modification. If servers are linked by DISTR, and each server has both `import` and and `export` specifications for each file being distributed, one can configure each server to export whatever it imports by writing

```
import.afterSuccess = sub {
  some('export.initiate');
};
```

The `export.initiate` method causes the server to invoke the appropriate `import` on other servers previously listed as clients of this one. Then, if configurations of servers are arranged so that one server updates all others, each server will independently update all its clients.

One must of course take care to insure that this distribution process does not create an infinite loop, either by design of the distribution topology or by insuring that loops otherwise terminate. E.g., one can prevent loops by insuring that `import.needed` aborts the exporting process when trying to update a file that is already up to date, as in RDIST. As with any recursive propagation technique, failure to insure this carefully can result in a network storm and network overload.

A safer scalable approach, though somewhat strange in semantics, is to write:

```
export.before = sub {
    some('import.initiate');
};
```

This causes servers exporting a file to synchronize with *their* servers before exporting it, so that the file gets propagated from some master server all the way down to the client. However, this can also cause version skews on the network. A request from a client causes a chained update of a *path* of servers between it and the master server, leaving other servers away from the path alone, even if they require updating. This

causes unpredictable version skews in server configurations if the servers use the file in their running configurations. The technique is entirely safe, however, if version skews are acceptable at leaf nodes, and if servers do not utilize distributed files in their own configurations. This is the case, e.g., in microcomputer networks where users are responsible for initiating their own updates.

Scaling Vulnerabilities

The security issues discussed above are even more important when DISTR is configured to broadcast files to a network. RDIST-like host-based authentication is vulnerable to spoofing attacks. DISTR's PGP authentication for imported files provides little help due to its vulnerability to replay attacks. A devious person can use these techniques to compromise a *network* configured for recursive propagation of files.

DISTR does not currently solve this problem. The two signed certificates of intent and time limits proposed above would only partially solve it. If a user makes a mistake in a file, signs it, distributes it, discovers the mistake, and redistributes it before the time limit on the original certificate, a devious person can set up a distribution chain for the incorrect file *in competition* with the corrected version.

What is Scalability?

Scalability typically refers to the way the performance of a software tool or algorithm changes as the number of computers it manages or utilizes increases without bound. While this definition is well suited to the needs of people analyzing network hardware or parallel computing algorithms, it is not so well suited to analyzing system administration tools. In algorithm analysis, the emphasis is on *performance* and *size*, while in system administration our emphasis is on *usability* and *complexity*. A usable tool must of course perform reasonably, and a large network is indeed a complex one. But system administrators typically have little interest in fast, unusable tools and may manage relatively small but relatively complex networks (such as my own).

For a system administrator, therefore, a scalable tool is one that handles problems with varying complexities and remains usable and cost effective at all scales of task complexity, whether or not complexity is caused by numbers of machines or environmental heterogeneity. A tool's scalability thus refers not only to its asymptotic performance on large networks, but also to its ease of use in performing small tasks, and the ease with which one can learn to perform small tasks with it.

My best example of a scalable configuration language is SLINK [8, 9], intended for managing images of local file repositories. Simple filesystem manipulations require only simple commands, while more complicated actions require more involved commands and a complete understanding of SLINK's protection model. In SLINK's case, the command syntax is crafted to discourage undesirable actions by making them more complex to specify [10]. In DISTR's case, instead, syntax is crafted so that homogeneous, domain-specific actions are easier to specify than heterogeneous, domain-bridging ones.

DISTR's language is designed to be the absolute minimum one needs for dealing with heterogeneous environments. Basic functions, including authentication, pre-processing, post-processing, and transport can be configured by specifying skeleton functions with a minimum of effort. Thus DISTR's *configuration language* is scalable in the sense of the above definition; easy to use for simple tasks and capable of performing any task with effort.

While DISTR's distribution strategy and language are scalable, its security mechanisms based on host address and PGP are not. Solving this problem will require mechanisms other than PGP.

How DISTR Works

To accomplish distribution actions, a DISTR daemon `distrd` runs on all hosts managed by DISTR. This daemon interprets a local configuration file `distr.conf`, reads requests off the network, and responds appropriately to each. Requests are made either by other daemons or by DISTR's front end user program `distr`.

Requests to the daemon contain not only the full name of the action requested (`import` on the remote host for exports, `export` for imports) but also everything known about the action on the requesting host, including all parameters specified for the action in the requesting host's configuration file and the contents of local files if appropriate. To transmit this data, DISTR uses a custom Perl library `Data::Pipeable` that can transmit any acyclic Perl reference structure through a pipe and reassemble a duplicate on the other end of the pipe. This library also allows DISTR to embed the contents of files of arbitrary length into the argument list sent to the remote machine.

`Data::Pipeable` is based upon the idea of the Comprehensive Perl Archive Network (CPAN) [16] library `Data::Storable`, which allows storage and retrieval of Perl data structures to and from disk files. Due to an improved algorithm, the new implementation does not utilize any C code to improve portability. Due to a lack of any sensible way to reconstruct them, as well as the potential security problems involved in trying, references to functions are not transmitted.

Both import and export requests are simple, stateless transactions consisting of a single request and a single response. Currently each `export` request contains the file's portable name and the target file's size, protection, owner, mode, and modification time. If these attributes do not match those on the server, the file is returned along with its own attributes and PGP signature if available. The returned file is then

checked locally for authenticity and installed if authentic. Each `import` request contains the file to be installed and all information known about it. This information includes the file's size, protection, owner, mode, modification time, and PGP signature if that is available. The remote client checks whether the file should be installed and installs it if installation is needed and allowed.

I have come to regret this design. I will soon be changing the `import` protocol to a two phase protocol (like RDIST's) with an initial probe for validity and a second phase in which the file is sent. This will not only save network bandwidth but also make the daemon more resistant to denial-of-service attacks where someone sends large unauthenticated requests to the daemon to keep it from processing valid requests. Currently invalid requests can fill filesystems with unauthenticated files.

DISTR's front-end program `distr` actually has a much more complex task than the daemon. Since the structure of inheritance defines an unlimited number of actions, the front end's job is to decide which actions to invoke. It does this in a very simple way, by looking at their attributes. An import event is meaningless without designation of a target file, while an export event is meaningless without a source file. When asked to import a file or files, the front end matches its request against all target files it knows about and imports only those targets. Likewise exports are performed only for defined source files.

Limitations

While DISTR may seem an advanced tool, it has many limitations imposed both by a need for simplicity and a lack of implementation time.

DISTR's daemon is inherently serial. It processes one request at a time, then looks for the next. This is both a limitation and a feature. While there is no compelling reason why the daemon cannot fork, I do not want it forking in my network! File distribution should never inhibit day to day operation of a workstation. Even if I do provide the daemon eventually with the ability to fork, I will limit it to at most four concurrent operations. This change, however, will change the semantics of DISTR. Currently, one can invoke serial updates on one host that are guaranteed to happen serially on all targets. This will not be so if the daemon can fork, and configuration files will have to be modified for this possibility.

The PGP implementation used in DISTR is version 2.6.2 using a variant of the front end written for PGP by Felix Gallo in implementing PENGUIN. This is a very limited implementation and provides the bare minimum of functionality. This is simply what was available at the time DISTR was written. I expect to replace this with a better PGP interface when possible.

DISTR's components, both client and daemon, scan the local configuration file as a text file to initialize themselves. Since this file contains Perl code, both of these compile that code on the fly. A lack of finesse in compiling each attribute's value results in quite cryptic error reporting, a deficiency I plan to remedy shortly. While DISTR in principle will execute on any host supporting Perl and daemons, it has only been tested for UNIX. Further development is needed to make it truly portable, including using improved PGP library functions.

While DISTR provides a transport layer suitable for communication between diverse hosts, and a portable namespace that hosts share, DISTR specifies nothing about exactly what names hosts will agree upon and what transformations will occur in translating information from one format to another. Semantics-preserving transformations, such as distributing UNIX groups for use in NT, must be hand-configured by the user.

Security

Despite my best efforts, DISTR is frightfully insecure. Unfortunately, DISTR's serious security problems are the same as would be encountered when implementing any scalable stateless security mechanism on a network. It is quite easy to compromise it with a replay attack.

For DISTR to be scalable, we cannot include mechanisms that compromise that scalability. This means that we cannot defeat replay attacks the easy way by assigning serial numbers to requests and enforcing simply increasing serial numbers. Each host would have to remember the last request serial number from each other host, and rebuilding a host after a crash (when DISTR is most useful) would be problematic.

Call me irresponsible, but DISTR's default running configuration is very insecure. The default authentication is host-based and RDIST-like, with all the security problems that implies, including being prone to address spoofing attacks. The reason for this is that the average user looking for something like RDIST will be more likely to utilize DISTR if it acts something like RDIST in the beginning, and less likely to utilize it if one has to learn everything about PGP first.

This has serious implications. If a naive user fails to heed my warnings and implements recursive propagation on a large network without PGP-authenticating the files at least, that user's whole *network* will be prone to spoofing. If that same user implements the PENGUIN lookalike code given above without using PGP, all bets are off on the security of the network. It is *very* easy to completely compromise the security of DISTR in the configuration file. Since DISTR never *enforces* security limits, but makes them optional, it can never be considered completely secure.

DISTR's daemon is also prone to denial of service attacks based on message volume. Since the

current protocol sends files to be distributed as part of the request, the daemon stores them in a temporary location before determining what to do with them. If a devious person sends very large files the daemon will happily fill up `/tmp`. Though it is more difficult to get DISTR to install those files anywhere crucial, this problem is very annoying and will be fixed by revising DISTR's protocol.

Critique

It is difficult to evaluate DISTR against other tools providing the same functions because DISTR's priorities are so different. Rather than configuring just a server, the user of DISTR must configure both servers and clients. This requires construction of one configuration file per *type* of client, as well as bootstrapping each client with the daemon and its configuration before using DISTR. Whether this is worth the effort depends upon one's goals. If the extra functionality of DISTR is appealing, the extra work, including the bootstrap, is probably worth the effort. If one desires RDIST functionality, then DISTR is much more work to deploy. If one desires remote privileged execution of selected scripts, DISTR's configuration provides the safest way I know to do it.

It is a dubious practice to provide a mechanism for use of standard naming without providing guidance about the naming standard, but this is all my time allows. To avoid massive confusion, users of DISTR have to remain consistent with their own naming standards. The best guidance I can give is to adopt a standard that works like the language, from general to specific, and clearly identifies platform dependencies by levels of the naming scheme. Everything under the name `unix` should refer to transactions specific to UNIX, while everything under `nt` should refer to transactions specific to Windows-NT. For cross-platform transfers, the name of the action should evoke the subsystem being updated, e.g., `groups` for a cross-platform transfer of user groups between different operating systems, `mail` for generic mail information, etc.

It is also a dubious practice to provide a modality for bridging domains without providing any utility functions to help. But I am ignorant of other users' needs. The namespace, however, provides an easy way to categorize and label server and client protocols for accomplishing various kinds of transfers. If DISTR or similar techniques prove popular, I hope users will help one another out in building these translations.

It is also questionable whether a distribution mechanism that requires hand-configuration of its distribution topology is scalable. In a large network, it is a lot of work to tell all the servers about all their clients and vice versa. But I know of no reliable automatic method for determining this information from lists of equivalence classes of clients. The general problem of mapping a distribution scheme optimally onto a given topology is intractable, and it is almost as difficult to precisely describe a network topology devoid of distribution topology as it is to describe the distribution topology itself. Worse, typically we would rather not allow a program to determine which machines are servers and which are clients; we decide which machines are servers beforehand. An alternative to explicitly hand coding the topology seems quite difficult to implement and of dubious value.

Conclusions

We should remember that in our capacities as system administrators we are not networking computers, but people. The first step in this task is to network the people who maintain the computers. But commonly, we ourselves do not manage to bind together into a group that is stronger than the sum of its parts. Lack of clarity in our roles combines with ambiguity in our powers to create situations that pit us against each other instead of against real problems with the network.

DISTR provides a beginning of a new camaraderie between administrators in what used to be opposing factions. There is no need to trust anyone with one's life – one can trust people to do what they do best and have no doubt that this is all they do. We no longer have to give an untrusted person the root password and wonder exactly how bad things can become. And we no longer have the potential to blame people for problems who could not possibly have caused them due to lack of privilege.

One of the most important parts of being a member of a family is learning to protect one's boundaries without excluding others. Through no fault of their own, fiefdoms play the role of outcast family members in a large family. Fiefdoms exist partially because they do not have the tools to protect their boundaries without excluding others. Tools like DISTR are the first step in transforming fiefdoms into families. Living in a good family is a lot more pleasant.

Availability

DISTR 2.0 is in testing and will be available by conference time in the directory ftp://ftp.eecs.tufts.edu/pub/distr. DISTR 1.0, although known by the same name, is quite different in function. It is a front-end to RDIST that implements only archiving and rollback features.

Acknowledgements

David Krumme endured multiple excruciating discussions of functionality during the development of DISTR's predecessors, and gave much valuable feedback and guidance. Scott Corzine gave me extensive advice and design help, and helped me develop the proper paranoic attitude about security and authentication. My spouse Elizabeth endured proofreading of many paper revisions. Special thanks to Remy Evard

for last minute help with clarity and references. The name DISTR is not just a pun on RDIST. DISTR is also the name of a primitive operator in the Berkeley implementation of Backus' functional programming language FP. FP's DISTR is also called the 'right distribution primitive'.

Author Biography

Alva L. Couch was born in Winston-Salem, North Carolina where he attended the North Carolina School of the Arts as a high school major in bassoon and contrabassoon performance. He received an S.B. in Architecture from M.I.T. in 1978, after which he worked for four years as a systems analyst and administrator at Harvard Medical School. Returning to school, he received an M.S. in Mathematics from Tufts in 1987, and a Ph.D. in Mathematics from Tufts in 1988. He became a member of the faculty of Tufts Department of Computer Science in the fall of 1988, and is currently an Associate Professor of Electrical Engineering and Computer Science at Tufts. In 1996 he received the Leibner Award for excellence in teaching and advising from Tufts. He has assisted in maintaining the Tufts computer systems for Computer Science teaching and research since 1985, when he was a Ph.D. student, and is currently responsible for maintaining the largest independent departmental computer network at Tufts. He can be reached by surface mail at the Department of Electrical Engineering and Computer Science, 161 College Avenue, Tufts University, Medford, MA 02155. He can be reached via electronic mail as couch@eecs.tufts.edu. His work phone is (781)627-3674.

References

[1] Jonathan Abbey, "opt_depot web site," http://www.arlut.utexas.edu/csd/opt_depot/opt_depot.html.

[2] Paul Albitz and Cricket Liu, *DNS and BIND, 2nd Edition*, O'Reilly and Assoc., 1996.

[3] Paul Anderson, "Towards a High-Level Machine Configuration System," *Proc. LISA-VIII*, 1994.

[4] David Bianco, Travis Priest, and David Cordner, "Cicero: a Package Installation System for an Integrated Computing Environment," http://ice-www.larc.nasa.gov/ICE/doc/Cicero/cicero.html.

[5] Murray Cole, *Algorithmic Skeletons: Structured Management of Parallel Computation*, MIT Press, 1989.

[6] Wallace Colyer and Walter Wong, "Depot: a Tool for Managing Software Environments," *Proc. LISA-VI*, 1992.

[7] Michael Cooper, "Overhauling Rdist for the '90's," *Proc. LISA-VI*, 1992.

[8] Alva Couch and Greg Owen, "Managing Large Software Repositories with SLINK," *Proc. SANS-95*, 1995.

[9] Alva Couch, *SLINK Manual*, 1996. http://www.eecs.tufts.edu/couch/slink.html.

[10] Alva Couch, "SLINK: Simple, Effective Filesystem Maintenance Abstractions for Community-Based Administration," *Proc. LISA-X*, 1996.

[11] David Flanagan, *Java in a Nutshell*, 2nd edition, O'Reilly and Assoc., 1997.

[12] David Flanagan, *JavaScript: the Definitive Guide, 2nd edition*, O'Reilly and Assoc., 1997.

[13] Felix Gallo, *Penguin-3.00*, available from CPAN [16].

[14] James Murray, *Windows-NT SNMP: Simple Network Management Protocol*, O'Reilly and Assoc, 1997.

[15] Kenneth Manheimer, Barry Warsaw, Stephen Clark, and Walter Rowe, "The Depot: a Framework for Sharing Software Installation Across Organizational and UNIX platform boundaries," *Proc. LISA-IV*, 1990.

[16] Jon Orwant, "Welcome to the Comprehensive Perl Archive Network!" http://www.perl.com/CPAN-local/CPAN.html, 1997.

[17] Pyzzo Software, Inc., "PC-Rdist Software Distribution System," http://www.pyzzo.com/pcrdist/.

[18] Rick Ramsey, *All About Administering NIS+*, Sun Microsystems Press.

[19] John P. Rouillard and Richard B. Martin, "Depot-Lite: A Mechanism for Managing Software" *Proc. LISA-VIII*, 1994.

[20] Hal Stern, *Managing NFS and NIS*, O'Reilly and Assoc., 1991.

[21] Larry Wall, Tom Christiansen, and Randall Schwartz, *Programming Perl, 2nd edition*, O'Reilly and Assoc., 1996.

[22] Walter C. Wong, "Local Disk Depot – Customizing the Software Environment" *Proc. LISA-VII*, 1993.

[23] Tatu Ylönen, "SSH (Secure Shell) Remote Login Program," http://www.cs.hut.fi/ssh/.

[24] "Lightweight Directory Access Protocol," http://www.umich.edu/ rsug/ldap/.

[25] "RevRdist Home Page from Purdue U," http://www.purdue.edu/revrdist/.

An Analysis of UNIX System Configuration

Rémy Evard – Argonne National Laboratory

ABSTRACT

Management of operating system configuration files files is an essential part of UNIX systems administration. It is particularly difficult in environments with a large number of computers.

This paper presents a study of UNIX configuration file management. It compares existing systems and tools from the literature, presents several case studies of configuration file management in practice, examines one site in depth, and makes numerous observations on the configuration process.

Introduction

Systems administration is hard, and is getting harder. This may be the computing world's single biggest problem. There are certainly others: security, privacy, improving performance, standards enforced by potential monopolies, the year 2000, etc.; the list can go on and on. But none of these matters if computers aren't useable in the first place.

In our modern distributed systems, each desktop is becoming increasingly more powerful and is expected to provide more and more functionality. Borrowing a metaphor from Rob Kolstad, the "service knob" is being cranked up, and systems administrators and users are paying by spending more time configuring systems, installing software, tuning networks, fighting fires, and trying to convince the environment to just work right. In the corporate world, this problem is usually referred to as part of the "total cost of ownership" or TCO, and it is a growing concern.

Simply stated – it is difficult to keep a computing system up to date and performing correctly. This has traditionally been the role of the systems administrator, and, as the requirements for computers continue to grow, the systems become more complex to administer. It is imperative that we make systems administration easier.

In computer science and engineering disciplines, complexity is often managed by abstraction. For example, source code is organized into functions, procedures, or objects with well-defined interfaces. Information is stored in data structures, allowing algorithms to be developed to manage abstract data structures. Abstraction methods are often used in systems administration as well. We often create a set of scripts or a tool for performing some particular function. As evidenced by the growing complexity in our field, we need to investigate more powerful abstraction mechanisms.

The work in this paper is part of an ongoing project to understand the underlying principles of systems administration. It is hoped that a deeper understanding will result in tools and methods that can be used to build stronger abstractions, and in new administration models that help to reduce the complexity of managing large and diverse sites of all different types.

The particular area discussed in this paper is that of operating system configuration files – the files in a UNIX system that control how the operating system and its constituent services perform. Classic examples are /etc/passwd, root's crontab file, and /etc/inetd.conf. The number of files configured on any particular system varies dramatically from one site to another and one architecture to another, but can range from a small handful to perhaps a hundred. Ultimately, these are the files that determine who can use the machine, how it can be used, and what takes place on it.

These configuration files are a good area of study because they are relatively simple but can lead to complex issues. They are quite well understood at the single-system level, but they require a very carefully planned strategy in a network of several thousand hosts. Each configuration file is a self-contained problem; but the files are typically grouped together, making them a choice candidate for an abstraction that encapsulates all configuration management in a system. In understanding how configuration files are created, managed, and distributed at a site, one will typically have to understand the site's management model (and, often, the political intricacies). In this way, configuration file study becomes a platform for understanding the other aspects of systems administration.

The goal of this study is to understand the operating system configuration process and the problems associated with it, to look at how different sites have approached this problem, and to consider various abstractions for managing the configurations of multiple hosts.

Although the problem of the complexity of systems administration spans all different types of computers, organizations, and management approaches,

this study was limited in scope in order to make it feasible. The discussions in this paper are principally applicable to heterogeneous networks of UNIX machines.

Configuration Management Background

Configuration file management is not a new topic to the systems administration community. Yet, despite multiple papers on the topic, there does not yet appear to be a commonly accepted approach to building new machines, configuring existing systems, or managing the files used in the process. While this may be a problem for systems administrators, it also means that there is a wealth of information from which to draw potential solutions.

As part of the background for this study, I spent some time reviewing the history of configuration systems. A detailed discussion of this review is in itself quite interesting but beyond the scope of this paper. A quick summary, however, may be help set the context of the study.

Interest and work in this topic dates at least as far back as the days of LISA I (all the way back to the Reagan years), when Ken Stone [Stone] presented a paper that described HP workstation disk cloning, making initial modifications with sed, and then performing later updates with rdist. Ironically, nearly the same method is used today in several very large sites.

Over the next several years, Sun Microsystems' NIS [NIS] became more widely used, due in part to the 1991 publication of the book *Managing NFS and NIS* by Hal Stern [Stern]. Other solutions from vendors appeared, including the Tivoli Systems *Management Environment* [Tivoli].

Several configuration systems and cloning scripts were detailed in various LISA proceedings. Then in 1994, in LISA VIII, the community nearly exploded with four configuration systems:

- Anderson's lcfg [Anderson]
- Harlander's GeNUAdmin [Harlander]
- Imazu's OMNICONF [Imazu]
- Rouillard and Martin's Config [Rouillard]

Each of these is quite different, but they share some interesting similarities. First, each grew out of a need for a more powerful tool than was currently available to the author. Second, each maintains a central description or database of the configurations that should be installed on individual hosts. I recommend that scholars in this area examine Anderson's paper for an excellent summary of the state of host configuration at this time.

In following years, configuration systems were used in increasingly sophisticated ways, or perhaps more accurately, were seriously discussed as a part of other processes for the first time. Shaddock and fellow authors [Shaddock] discussed a use of their sasify system to do a massive upgrade of 1500 workstations. Fisk [Fisk] the rather hazy barrier between machine configuration and software distribution, and described a system that tackled both areas as part of the same problem.

The general approach taken by the administrative community over this time period has been to develop a host cloning process and then to distribute updates directly to hosts from a central repository. The diversity of solutions developed illustrates that this is a basic problem for many sites with – as is not surprising – a wide range of requirements.

Site Case Studies

During the past two years, I moved from a site where we had rigorous configuration management to a site that had ad-hoc methods of keeping machines up to date with good informal methods but no formal structures in place. The difference between the two sites struck me as remarkable. This was one of my primary motivations for examining configuration files in detail.

Initially, I thought that my new site would be much more difficult to manage at the host level, requiring a lot more hands-on management, but that was usually not the case. Instead, the differences were really about how easy it was to manage the entire environment.

At the first site, it was easier to delegate management of machines to different people, because no single person had the configuration of an architecture in their head: it was all kept in the central configuration files and build scripts. Global changes such as an inetd replacement or a new shell could be easily performed, and so they often were, making for a rich environment.

On the other hand, at my new site, it was much simpler to handle new architectures, because there was no overhead in assimilating them into a global system. One simply set up the machine, tweaked it until it worked, warned new users that it had a minimal environment, and then left it alone. This resulted in more flexibility at the host level and less in the larger environment.

Intrigued by these differences, I started to talk to administrators at other sites to learn how they handle configuration management. During the past year, I've talked about the issues with approximately thirty different groups. These studies were informal, usually occurring as a series of conversations on the phone, around a whiteboard, or over lunch.

I present a summary of a number of these discussions here in order to impart a general idea of the range of the sites and strategies. These sites are not intended to be representative of the industry as a whole; a far larger study would be required for that. Instead, they provide insight into how other sites do configuration management, and the general state of systems administration at a number of different sites.

All sites and participants have been kept anonymous except for Northeastern University. I worked there and played a large role in the design of its systems, and feel that I should acknowledge my own role in the evaluation of its environment.

Case Study 1 – Northeastern University

The College of Computer Science at Northeastern University runs a network of approximately 70 Suns, 40 Alphas, 50 PCs running Windows variants, 50 Macintoshes, and a number of special purpose UNIX machines. These are managed by a central administration group that is responsible for all aspects of the technical environment.

At NU, a new machine is built by installing an operating system from media, following a set of instructions to configure it, and then applying modifications from the network. NIS is used to coordinate most of the files that it can support. All other configuration files are maintained in a central location under RCS. The configuration directory is NFS exported to all hosts. Machines are updated manually by using a homegrown system based on a root rsh mechanism from a central server that then installs the correct file onto that host. A number of tools have been built around this mechanism to automate the distribution of files. In general, changes to local machines are kept to a minimum through this mechanism, even though several machines have very different configurations from others (in part because the central repository is able to store different configurations for different machines).

When the system was first installed, it solved a number of important problems. Over the years, the environment has grown more complicated. The administrators have identified new requirements for the system, such as keeping changes to the OS and other configuration information on all client machines in sync, even when machines are down temporarily. This is especially important to them in order to keep all machines current the latest vendor security patches. They expect to completely rework the system soon.

Case Study 2

Site 2 is a computer science department with about 70 UNIX-based computers. The majority of these are dataless machines (with just swap on the local disk) that get their filesystems over the network from an Auspex NFS server. The remaining computers are SGIs, and can't boot from the Auspex, so use their own local disks and a centralized /usr/local-like scheme.

If a machine is a client of the Auspex, building is pretty simple: an additional set of directories is created for that machine, and it is configured to netboot from the Auspex. Changes to those machines' configurations are done on the Auspex, by editing the file directly on the Auspex's file system and then copying changes to the other clients. RCS is used for change management of key system files.

The other machines in the department are set up individually, each by hand. If changes need to take place to them, the administrator logs in and makes those changes. There is some expectation that this method won't scale to large numbers of machines, but that's not seen as an important issue at this time.

Case Study 3

Site 3 is a large Fortune 100 corporation. There are many groups within the company who are responsible for different parts of the infrastructure. The particular group that was interviewed is responsible for the environment for a large development and engineering segment of the company. The set of machines that they are responsible for includes 1000 Sun workstations and 5000 X terminals. In addition, some people in the group have responsibility for other architectures within the company including HPs and SGIs. The approach to managing these other computers is completely different than the management of the Suns, and was not discussed during the interview due to a lack of time. The group is responsible for the operating system and the applications on the Suns, but does not manage the network, or some networked applications like email. Their users and the machines they manage are in multiple locations spread around the world.

This group has divided their computers into small modules, each consisting of a machine for general logins, an application server, and a number of compute servers. Users are associated with one particular module and use X-terminals to access the resources.

The machine configurations are kept on a set of master hard drives. New machines are cloned from these hard drives, and then the initial boot-up script asks a series of questions in order to initialize the host. The master configurations are rigorously maintained, with files documenting all changes kept in critical directories.

NIS is used to distribute password, group, and netgroup information. In order to scale NIS to their environment, the group rewrote the NIS transfer mechanism to introduce another layer of hierarchy. Some files, such as /etc/passwd, are pushed out using an rdist mechanism, while other files, such as /etc/printcap, are maintained largely by hand and distributed by complex scripts to each system architecture. OS patches are kept on designated OS masters, with changes tracked in RCS control files. These patches are distributed using a combination of rdist and ftp.

In some cases, the group has to do direct hands-on management. Notably, the administration group had to install the 5000 X-terminals and configure them by hand because of security concerns and an SNMP bug.

The group is having troubles with the disk cloning strategy because of an increasing number of variables: operating system versions, different sizes of disks, different types of computers, and, most importantly, organizational changes.

Case Study 4

Site 4 is a growing company currently expanding to multiple campuses. The primary UNIX computer users are engineers using CAD applications. A central administration group is responsible for all aspects of managing the computing infrastructure, and is divided into several different groups with separate areas of authority, such as UNIX, PC desktops, and networking. The environment consists of 1500 Intel-based PCs and 900 UNIX machines, most of which are Suns running Solaris 2.5.x. There are also a dozen HPs and SGIs that are managed independently by specialists within the UNIX group.

The Suns are built by using Sun's JumpStart [JumpStart], which solves the build and initialization issues. The group uses NIS to manage password and other changes. Further configuration of machines almost never takes place, due to a very strong emphasis on centralized servers. When changes do need to take place, they are pushed out from a server using a script wrapped around rdist, which takes advantage of clever hostnaming conventions in order to make decisions about what hosts to affect. Central files are not kept under revision control, but backup copies of critical files will typically be maintained.

The group uses approximately 30 servers to support the 900 Suns. Those servers are managed in a more ad-hoc way, with a lot of hands-on modification of configurations, primarily because the servers span a wide range of services and hardware.

Interestingly, the smaller HP and SGI environments are managed in a much looser way, with individual host configuration typically taking place directly on the host. Thus, the centrally managed approach of the Suns comes from a need to manage on a large-scale, not from a mandate from management.

The group anticipates that the next operating system upgrade may be very difficult and, despite the fact that machines are well behaved in this system, is nervous that things are on the verge of getting complicated.

Case Study 5

Site 5 is a small university department serving a combination of computer science and art graduate students. Their network consists of some thirty SGIs, a couple of Suns, and a scattering of Intel-based and Macintosh personal computers. The direction of the infrastructure is determined almost entirely on the availability of funding and the need for project development and demos.

Each of the SGIs is generally built from CD-ROM or by doing a byte-for-byte copy of the system disk of a previously built system. Since performance for demos is a big concern, patches are applied very sparingly, and considerable work is done to verify that the vendor patches do not break or slow down existing code.

Each machine is individually maintained by hand. This approach is taken to avoid having a central machine that, if compromised, would allow for easy compromise of others. In this dynamic university environment, security is a big issue. Each systems administrator has an individual root account in /etc/passwd on a given machine. Various people in the environment, beyond the system administrator, have root access to selected machines in order to facilitate research and development by installing software, changing kernel configurations, and permissions for /dev devices.

NIS is used to allow department-wide logins. The system administrators of this network control their NIS maps, but send email to another group for updates to their DNS tables. It is felt that the time required to set up a DNS server would be better spent on immediate pressing issues.

Any systems other than SGI are maintained fitfully or not at all; attention is given to them only in the case of a particular user need or security incident.

Backups of system areas of critical machines are performed, but users are expected to back up their own files as the user deems necessary. DAT tape drives are provided in public areas for this purpose. There is no change management for configuration files; copies of relevant files can usually be found on similarly configured machines.

The administrator of this environment is well past the point where he can keep up with all of the changes that need to take place.

Case Study 6

Site 6 is a financial company that is spread across several cities. As with many large sites, the infrastructure is managed by several different groups, who are divided both according to function (i.e., networking) and according to company directions (i.e., all activity based around one type of interaction with clients). The focus of this study was a part of their computing infrastructure used to build, maintain, and run one particular application, where uptime during business hours is the prime directive. This environment consists of about 350 Suns, all running Solaris. 200 of these are used for running the application, 100 of these are development and support machines, and the remainder are servers of various types.

NIS is used within this environment to deliver passwords, automount maps, and some special maps used by administrative applications. NFS is barely used, because of the importance of minimizing dependencies.

The application machines are critical and are carefully controlled. They are built either from JumpStart or from a cloned disk that boots up into an interactive initialize phase. The developer machines are less carefully managed, and will typically be built by hand. The servers run on a number of different types of Sun hardware and have all been custom-built. The group uses an internal web page to maintain a checklist of things that should be done when building a machine. Over the past year, one of the group's major projects has been to get the servers and the developer's machines "rationalized" or similarly configured.

Each machine has a separate root password, and there is no centrally authoritative machine. However, the group uses a "master root cron" mechanism to achieve the same effect. Every half hour, the cron job checks to see whether there is a new crontab available on any of several replicated NFS servers. If so, it is copied in as the new crontab, which is, of course, executed as root. The group uses this mechanism to install carefully crafted patches, to update configuration files, and to make global changes as necessary.

The group is pretty happy with their system. Other than the dependence on NFS for some central functions, the environment is quite failsafe and reliable. There is some dissatisfaction with the server and development environments, but those are being fixed during the reconciliation process. The hardest problem they have is finding all the machines.

Case Study 7

Site 7 is a research lab with an emphasis on computational science. The infrastructure consists of several supercomputers, a UNIX-based workstation network with over 100 UNIX machines of many different types, a growing number of PCs, and a production network based on ethernet and ATM. Most of the infrastructure is managed by a central group, with some of the experimental labs being managed by individuals focused in that area. This study focused on the machines managed by the central group.

There is one NIS server for the department, and all machines are a member of the NIS domain. NFS is the primary remote file system in use, although AFS and DFS are used minimally. The build process for a new machine depends on what type of computer is being built, but the group is working to standardize methodology. Typically, one will install the operating system onto a machine from CD-ROM, then follow written instructions to get the machine onto the network. After that, a script applies relevant patches and makes changes to the local machine.

Many changes are handled through NIS, but occasionally changes must be pushed out to all machines. When this happens, the group generates a list of machines and then does an rsh from a central server to push out the changes. Until recently, no precautions were taken to check for machines that were down, or to use revision control on the sources of the files. Some of the machines in the environment are special purpose or specially configured, and the set of machines is constantly moving and being reconfigured, so a hands-on approach was the simplest to develop.

This approach resulted in a somewhat inconsistent environment and was too difficult to use for all but the most serious modifications, so individual hosts weren't tuned often to match new requirements. When they were updated, the build scripts weren't necessarily changed to reflect that update, so machines built after the change might or might not have that new change.

The group is moving to a centrally managed set of configuration files, and a standard mechanism for installing new hosts based on these files and centralized sets of OS patches. There are two main concerns with this system: first, it must support individual machine idiosyncrasies, and second, it must be able to handle machines that are down or disconnected from the network.

Case Study 8

Site 8 is an engineering department in a university. A lot of the administration work is done by students, and the policies and procedures reflect this. The large environment consists of many different types of UNIX machines, including BSDI, NetBSD, Solaris, SunOS, Alphas, HPs, and some Windows boxes and Macintoshes added for flavor.

Many machines are built by students by hand. Others are built by doing a network boot and then getting the latest set of modifications.

A set of HPs is used for most of the central management. The HPs use both NIS and rdist to distribute files into the environment. In many cases, the source files are built by using either Perl or m4 macros, because the environment is complicated enough that the source files are hairy. The rdist files are built using gnumake, and the source files are kept under RCS. They've found that, because of the number of new students they work with, detailed logging is important.

This is a rather complicated system, and one of the most difficult tasks is to incorporate new architectures into it. The administrators would also like the ability to put comments and documentation within the source of files, and feel the need for a comprehensive database of hosts.

Case Study 9

Site 9 is a research lab with a focus on computer science. The UNIX environment consists of about fifty DEC Alphas running Digital UNIX, along with a few SGIs.

Builds of new machines are done by using a customized version of the Digital UNIX install process. It builds the local machine, makes some modifications, and then invokes an rdist on the machine to add files

from a central collection. The administrators can build a number of machines simultaneously, spending only about five minutes per machine. The entire process takes about two hours.

The site uses a rdist system to manage configurations on all of the machines. It is used to push out aliases, fstab, automount files, printcap, and others. The rdist scripts are run nightly, and not invoked directly by the administrator. Maintaining the list of target hosts for rdist is one of the bigger problems. The files that are pushed out are generally maintained by hand, although some of them have special rules that are applied on distribution. For example, fstab incorporates any fstab.local it finds on the target machine.

The site does not use NIS, so all password changes must take place on a central machine. New accounts and password changes are pushed out using the rdist system.

The administrators aren't particularly happy with the mechanism, although it works. Among other things, they would like to see a pull mechanism rather than a centralized push. The system has been in place for quite some time, and given the staffing levels, they are unlikely to be able to change it for some time.

Case Study Observations

As I mentioned above, this sample set is too small to generalize to the entire industry. Nonetheless, a number of interesting observations can be made, some of which may help to understand what is needed in a stronger abstraction method.

- Almost every site uses NIS, although some use it to distribute for only a few maps, while others used it for every map intended.
- No site uses NIS+, not even the Sun-only sites.
- No one seems to settle into a definitive way of doing things on every host. Most sites have more than one method that they use for building machines and more than one method to configure them. Site 6 is a good example of this; they use JumpStart in some cases and disk cloning in others (once they even participated in a race between two administrators who favored different approaches). Site 7, while having a build script for some architectures, doesn't have a build script for others. some architectures, didn't have a build script for others.
- The large sites typically have a very controlled method for managing most of their machines, and a more ad-hoc method of managing their servers. The way they manage their thirty servers often is similar to how a thirty-machine site manages its entire environment. This fact has some very interesting ramifications.
- Centralized management and automated building of some type or another are done at nearly every site with fifty or more machines. In the cases where this isn't true, building is done by giving a cd-rom and a set of instructions to a student (which is nearly the same as automated building).
- Once a site has a strategy, it is stuck. Whether the staff have invested heavily in one vendor's build mechanism (like JumpStart), or an Auspex, or a management scheme, they find it very difficult to move beyond the restrictions imposed by that scheme. This is one reason that a major OS upgrade or the installation of a new architecture is so hard. Not only must the administrators learn the nuances of the new system, they have to modify their existing practices in order to support it.
- At some sites, changes take place constantly, at others rarely. This appears to be a function of a number of variables: how comprehensive the build process is, the ways in which the machines are used, and the need for the environment to stay modern. Schools and research labs seem to have more dynamic environments, while corporations seem to focus on supporting one type of application and then not changing once the application works well.
- A surprising number of people feel that keeping track of machines is the hardest problem they have with configuration. If a complete list of machines could be generated, it would be much easier to keep them up to date.

Site	Environment	Build	Configure	Revision
1	100 various	Media + Script	NIS, file push	RCS
2	70 dataless Suns	Copying	NIS, edits	RCS
3	1000 Suns	Disk clones	NIS, rdist	-
4	900 Suns	JumpStart	NIS, rdist	.bak
5	30 SGI	Media	NIS, edits	-
6	350 Suns	JumpStart / Clone	NIS, cron copies	-
7	100 various	Media + Script	NIS, edits	-
8	100 various	Media + Script	NIS, rdist	RCS
9	50 Alpha	Digital UNIX install	rdist	RCS

Figure 1: Environment attributes.

- Some sites differentiate between the build process and the configure process. Others don't touch a machine after building it, except in extreme cases, while still others don't have any more formal build process than a set of notes from the last time they did it. Again, this comes down to what the computers are being used for.
- Everyone in the survey who has an update system uses a push mechanism. In some cases, the hosts pull down the files, but that pull is initiated from some central spot. No one is doing an explicit pull, where the action on the host is initiated by the host or the user on the host. (This may come from the fact that I always spoke with an administrator who was part of some kind of central support organization, not with a user who was responsible for their own machine.)

Also, a few notes of non-technical nature:

- Everyone has a different definition of what "server" means.
- Nearly everyone feels overworked and said something similar to "I've been too busy to take the time to go back and fix that."
- If you've seen one machine room, you've seen them all. But it's still a lot of fun to see them all.

An In-Depth Look at One Site

For four years, Northeastern University's College of Computer Science (Site 1 in the above section) has been using a central configuration mechanism to manage most of its files. I have studied the files in this system in some depth in order to understand what was being changed and how often those changes took place.

A bit of background on the NU configuration system will be helpful. The system is based on a central NFS repository, where all UNIX machines, regardless of architecture, retrieve their files. Multiple copies of a single type of file can be kept, with specifications based on hostname and architecture type. So, for example, if a sun4 named "sol" were to look for a

File	Different Versions	Revisions	Type	File	Different Versions	Revisions	Type
amd	2	1-2	admin tool	svc.conf	2	1	OS
cops.cf	3	1	admin tool	syslog.conf	8	1-7	OS
etherdown	1	1	admin tool	termcap	2	3	OS
newsyslog	2	1-3	admin tool	ttys	2	1	OS
rotlogs	1	1	admin tool	ttytab	10	1-2	OS
staticroutes	1	1	admin tool	rc	5	2-3	OS bootup
sudoers	1	1	admin tool	rc.local	11	4-14	OS bootup
super-users	3	50	admin tool	rc.priv	25	1-17	OS bootup
watchmerc	11	1-4	admin tool	hosts.lpd	1	1	printer
crontab	7	1-4	cron related	printcap	10	1-10	printer
daily	16	2-26	cron related	aliases	1	2	service
hourly	5	3-11	cron related	ftpusers	1	1	service
monthly	4	1	cron related	hosts.allow	9	6-16	service
weekly	10	2-11	cron related	hosts.deny	6	3-6	service
bootparams	1	1	OS	lbcd	1	1	service
bootptab	1	5	OS	mrouted.conf	4	1-6	service
exports	3	1	OS	ntp.conf	4	3	service
format.dat	2	2/3	OS	resolv.conf	6	1-4	service
fstab	1	1	OS	sendmail	2	1	service
group	4	1-2	OS	sendmail.cf	2	2	service
hosts.equiv	1	1	OS	zshenv	1	5	shell
inetd.conf	11	4-15	OS	profile	1	5	shell
magic	1	1	OS	profile.bash	1	4	shell
nis	1	6	OS	tcsh.cshrc	1	6	shell
passwd	45	1-10	OS	Xconfig	1	1	X config
securenets	1	3	OS	xlogin	1	1	X config
securettys	3	1	OS	Xsession	1	1	X config
services	4	1-2	OS	Xsetup	1	1	X config
shells	2	1-3	OS	Xstartup	1	1	X config

Figure 2: Northeastern's config files.

passwd file, it would first select the file "passwd.sol" if it existed. If not, it would select "passwd.sun4." If that didn't exist, it would copy "passwd." This mechanism allows the administrators to set up defaults for the system, override those for specific architectures, and then override those for individual hosts. Thus, if one file will suffice for the entire system, there will only be one copy of it in the repository.

I've grouped these files by their function as I perceive them. For each file, I've noted two pieces of data:

- "Versions" is the number of different copies of that file are kept in the repository (so for the above example there would be three copies of the passwd file).
- "Revisions" is the number of times that file has been modified during the last four years. Because each version of the file might have multiple revisions, I've given the range of revisions.

The files are listed in Figure 2.

Some of the entries in the figure require some explanation or deserve some comment:

- After five revisions, bootptab has moved out of the config system because it is being autogenerated from a database of hosts.
- exports and fstab are in the repository but aren't actually distributed by the system. Instead, these are managed by hand on all hosts.
- group is the copy of /etc/group with NIS hooks in it.
- passwd has an enormous number of different versions. All differences amount to which netgroups are in which files, since this environment has restrictions over who can log in to which machines. Even with this number of files, it is possible to change the root password everywhere by running a sed script to change all of these files, and then typing "pushfile /etc/passwd".
- aliases and ftpusers are no longer used, since they are maintained as part of the central mail and ftp servers.
- hosts.allow and hosts.deny are part of the tcpd program, which controls access to various ports. These files have been changed extensively.
- sendmail is the actual sendmail binary. This is an interesting change for the config system, which, other than this file, is used only for ASCII files.
- amd is a script used to startup and shutdown the amd automounter.
- etherdown, rotlogs, and staticroutes are homegrown utilities.
- super-users is a file used to list who can have super-user access on a machine. At one point, there were many more versions of this file than just three, but the differing copies were recently eliminated.
- hourly, daily, weekly, and monthly are scripts executed by the root crontab at the frequencies you would expect. These are typically used to do maintenance on various kinds of servers, such as rotating logs, cleaning up tmp, and so on. The high number of daily and weekly files reflects the number of machines running customized services.
- rc, rc.local are used on suns to replace /etc/rc and /etc/rc.local.
- rc.priv is an augmentation of /etc/rc.local that is typically host specific, often used to start services on that particular machine such as a web server. Again, the high number of these reflects the number of machines running customized services.
- tcsh.cshrc, profile, profile.bash, and zshenv are the central files used to control all of the shells in the environment. These are changed only to make major changes to the default environment. Most PATH changes and other modifications are set using a different system.
- The various X* files are part of the Alpha CDE environment, and have a small number of revisions because they are newly added to the repository.

From this information, it is possible to make some statements about the role of the configuration management scheme at Northeastern. It is not clear how many of these observations will be relevant to other sites, but they nonetheless provide some insight into what kinds of patterns may be observed in the configuration of a network.

- A file that only has one copy and a small number of revisions is typically a file that was shipped with the OS but changed just a bit. The build process copies in the changed version of that file.
- The repository is used to change existing OS files, to add new files to the OS, to distribute a binary, to distribute scripts, and to distribute local configuration files for various processes. This is more than a simple configuration mechanism, but less than a complete solution, since it doesn't really handle software distribution, operating system patches, and other types of modifications. The system seems to have evolved into a mechanism for pushing out "things on individual machines that change or need to be kept locally."
- A number of files in the repository aren't distributed to hosts anymore, and some never were.
- It is not possible to gather this information from the table, but is worth mentioning that over twenty different people made modifications to the files. The system uses RCS for change

management, and this seems to have worked.

- A very small number of files had changes that were made and then reversed. RCS was used to detect a problem and move back to the last known good configuration. This does not seem to have been a common occurrence.
- At a first glance, it's difficult to generalize the number of times a file is likely to be changed. However, there are some distinct patterns. Files like super-users and passwd were changed quite often, due primarily because of changing roles of users. New students would arrive and be given root privileges, new machines were purchased on grants and had to number of people who could use them had to be constrained. In these cases, the files have some direct relationship to the role of the people who use the computers, so they changed more often than files that didn't.
- Another type of file that changed relatively often were those relating to some aspect of the physical environment, such as printcap, which had to be updated every time a new printer was purchased or an existing printer was moved.
- Files relating to the purposes of a specific machine, notably inetd.conf and the daily cron jobs, tended to have a high number of different versions, with some of those files having a high number of revisions.
- For the most part, the number of times that files change over the course of four years is pretty small. This may imply that emphasis should be put into the process of building a machine and getting it up to date with respect to the rest of the environment, rather than working on the process of distributing files.
- Some files aren't in this repository at all, or were in it but then their functionality moved away. This reflects a move toward centralizing a service, or encapsulating the functionality of the file into a different system. For example, sendmail.cf and aliases were minimally updated because major changes took place on the central mail hub. (In contrast, the aliases file on the hub has been updated 961 times during the same period.) Files like group, passwd and bootparams are generally updated via NIS. The central files for the shells are only updated minimally because they use another systems to set global paths (which has been updated 62 times). /etc/motd is not updated via the configuration system because it is never changed. Instead, the group uses a msgs-like system to make announcements. In every case, the move to a centralized system made the function provided by that file easier to manage and support.

Finally, to complete the story, the primary maps that are distributed via NIS at Northeastern are listed in Figure 3. In this figure, the "revisions" column also refers to the number of times the file has been updated since 1993. A few entries are worth describing in detail.

- The amd.* files refer to various automount maps used by the AMD automounter.
- The amd.home file has a line in it for each user of the system, specifying the mapping of their home directory.
- The amd.net and amd.home are actually in their

File	Versions
amd.ftp	5
amd.home	3241
amd.net	132
amd.proj	127
archtree	4
bootparams	3
ethers	2
group	415
hosts	4
netgroup	2000+
netgroups.aux	3
netmasks	1
networks	1
passwd	3913
protocols	1
publickey	1
rpc	3
services	22
ypservers	38

Figure 3: Primary maps at Northeastern.

second and third major releases, using a feature of RCS that lets one change major version numbers. This facility was used to mark major changes in the environment. The revision number given is a sum of all changes in all releases.

- The netgroup file is created using a script that collates several source files, some of which come from a database. The actual number of revisions is very difficult to calculate, but the number of revisions to the source files is approximately 2000.
- The hosts map is not used to distribute hosts information. It's essentially empty. All name lookups are done via DNS, which is largely independent of the hosts map. The database that is used to build DNS files has been changed 502 times in the last two years, and probably 1000 times in the last four.

Again, a number of observations specific to Northeastern can be made:

- There are several orders of magnitude difference between some of these maps and others, and several of the maps have been changed far more often than any files in the configuration system. It's clear from the number of changes in the passwd and amd.home maps that NIS is being used to support those files in the environment that change the most.
- For the most part, the files can be categorized by order of magnitude of their revisions.
  - O(1000) files include amd.home, passwd and netgroup. These are the files that must be changed in order to create a new account and arrange for it to be usable. These files reflect daily changes in the system, and they are very tightly coupled with the users of the environment.
  - O(100) files include amd.net, amd.proj, and group. The amd files are changed when the group adds new disks to some computer somewhere and must make them visible to the environment. The group files are changed most often when students enter or leave groups. These correspond with the super-users and password file changes in the central configuration system. These files reflect ways in which different parts of the organization use the environment.
  - O(10) files include services and ypservers. These change when some new service or function needs to be added to the environment, or when the network structure is modified. They correspond with centrally managed files like inetd.conf, rc.priv, and daily. Changes in these files reflect modifications to the network and to machines used as servers, and indicate a new function or a major architectural change in the network.
  - O(1) files are generally either unused or setup once and then forgotten, reflecting some part of the environment that rarely changes. Ethers, for example, never changes because this site does no diskless booting. Further changes typically reflect minor fixes. This pattern is generally true with the centrally managed files as well, although in some cases, such as the shells, the small number of modifications is because the mechanism for change has been delegated to another part of the system.
  - Some files do not fit perfectly into this ranking. For example, the hosts database has been updated around 1000 times, making it an O(1000) file. The hosts database at this site is comprehensive, including information such as architecture type, IP address, user, and OS version. Due to the expansive nature of this

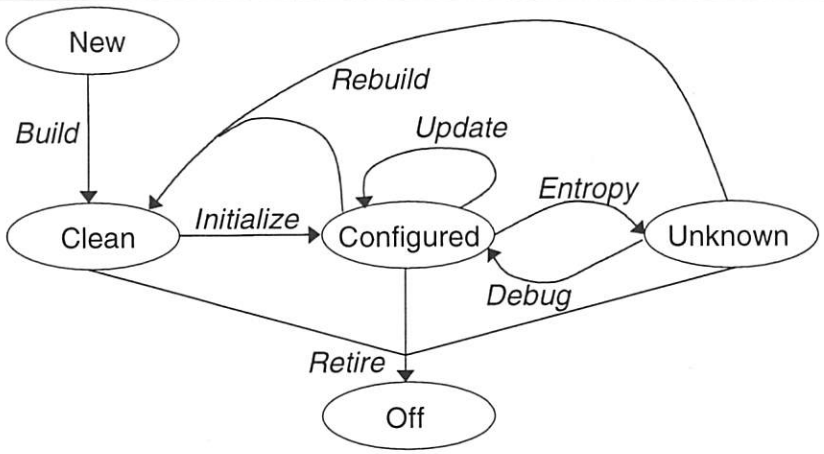

Figure 4: The lifecycle of a machine.

information, it is hard to generalize when changes are made to the hosts database, other than to say that it is changed whenever something relating to the identity of a machine is updated. More research into this area is required.

Keeping in mind that this data spans four years, the actual number of revisions of files is less important than how often they change relative to each other.

- It is possible for the password files and the amd.home files to be maintained as actual files on every host. This would require that new files be pushed out on the average of three times a day, which would pose several problems for the configuration system, including delivery to all machines (including ones that are down) and speed of distribution. In general, the NIS mechanism scales better to a high frequency of changes than their existing configuration system.
- The NIS system has achieved a certain degree of abstraction at this site. The administrators do not think of "adding a new user to the entire network by changing a file in NIS," they simply think of "adding a new user to the NIS maps."

Whether these statistics and observations will correlate with those of other sites remains to be seen. Regardless, it is hoped that this data will provide some insight into where and how often changes are made in a real-world environment.

General Observations and Theories

The real killer for systems administrators is change. Requirements change, new versions of applications appear, hardware becomes outdated, users change their mind about what they need . . . everything's a moving target. A configuration management system, in part, is the process by which an administrator controls and manages operating system and host changes.

From the literature survey, the site interviews, and the in-depth study of Northeastern's configuration file changes, I have collected a lot of observations and developed a few conjectures about the patterns of changes that take place on a host and in a network of computers. It is important to understand these patterns because they can be helpful in understanding the issues and requirements in a configuration system. Furthermore, they can be of help in developing stronger models and abstractions that may eventually result in an improvement in systems administration methods.

Changes Within A Machine Life Cycle

The life cycle of an individual machine is an interesting place to investigate the role of change. This cycle in itself is a complicated process and worthy of further study, but not in this paper. A sufficiently detailed version of the cycle is given in Figure 4. In the figure, a machine moves between these states:

- New. A new machine.
- Clean. A computer with the OS installed, but not configured to work in the environment. (For example, its network is unconfigured.)
- Configured. A computer that is configured correctly according to the requirements of the computing environment.
- Unknown. A computer that has been misconfigured, or has gotten out of date, or perhaps been borrowed by an intern and returned with stains on it.
- Off. All done. Retired. Excessed. A dead parrot.

The interesting part of the figure are the changes that a machine goes through as it moves between states. These are "processes," and consist of:

- Build. During the build process, the operating system is installed on the machine.
- Initialize. This occurs directly after build, and at many sites, is thought of as part of the same process. This is the initial set of modifications to the OS image that are required to have the computer operate in the environment. This will typically include network configuration, and may include OS patches and other changes. Once initialization is complete, the computer is (theoretically) a functional citizen of the computing environment.
- Update. At some point after the initialization, the computer will probably have to be modified. Perhaps the network configuration has changed, or a user needs to be added, or an OS patch needs to be applied, or the machine needs some kind of new functionality. Whatever the cause, the computer needs to be updated in order to bring the machine into conformance with the requirements. In most cases, this will happen continually for the lifetime of the computer.
- Entropy. This refers to the gradual process of change that results in a computer that has an unknown state. The causes for this are numerous; they include, for example, undisciplined changes made to the machine, major changes in the environment, or unexplained problems.
- Debug. This refers to the process of debugging an "unknown" machine, and getting it back into spec. This is usually an intensive, hands-on experience. Debugging can often involve updating as well.
- Rebuild. In some cases, a machine will need to be rebuilt, either because of some kind of problem or because the changes to be made are so drastric that simple updates make no sense. For example, a rebuild is typically done when upgrading from one major revision of an OS to

the next. The rebuild process usually consists simply of reapplying the build and initialization processes to the machine.

- Retire. This is the process of turning a machine off. In some sites, there is an official process for this, in others, it merely involves turning the computer off or forgetting it exists.

Obviously, these are generalizations. In some sites, and for some operating systems, the "build" and "initialize" processes are the same thing. At other sites, there is no particular definition that can be pointed at to say that a machine is "configured." It may only be "working right" or "broken," which, in effect, are the same thing as "configured" and "unknown."

In fact, this may be an important point. The literature and common knowledge implies that there is some strong definition of how a machine should be configured in an environment. Yet the majority of the sites that were interviewed could not say with certainty whether or not any of their hosts matched that definition; they could only say that they were working without complaint. It may be that we need a less rigorous concept of an environment definition.

In this life cycle, the desired state of a computer is the "configured" state, and almost all of the effort in the life cycle involves trying to get a machine configured and keeping it there. Changes to a machine only take place in the process portion of the diagram. So why do these processes take place, and what do they entail?

The "build" and "initialize" processes take place because a machine is in a known state (new or clean) and must be brought into conformance with the definition of a configured machine in the environment. Here the machine needs to be updated, but the requirements are stable.

The "update" process takes place because a new requirement has been identified, and the machine needs to reflect that requirement. In this case, the machine needs to be updated because the environment has changed.

The "entropy" process is perhaps the most interesting and least understood, and is the section of the graph that needs the most expansion. For our purposes, entropy includes any kind of situation where a machine is discovered to not have the correct configuration, and getting it back to where it should be will be difficult.

The "debug" process can be painful and time-intensive. It takes place only because entropy has occurred. (This is different from debugging a known configuration.)

Finally, the "rebuild" process takes place either as an alternative to debugging, or because the changes that must take place are so extensive that the "initialize" process is preferred. The rebuild process is a way of taking a machine in any state and moving into a known and understood state. It takes place either because of massive changes in the environment or massive changes in the machine.

In general, changes must take place on a machine for these reasons:

- The machine needs to conform to the environment.
- The environment has changed, and a machine needs to be modified to match it.

Using this model, we can identify the high-level requirements for an abstraction mechanism to manage configurations:

- It must contain or have access to the definition of the environment.
- It must be able to perform or replace each of the processes that result in a configured machine, i.e.:
  - build
  - initialize
  - update
  - debug

As noted above, all of the effort in the life cycle is involved in getting a machine into the "configured'" state. It may be worth considering a model in which the configuration abstraction simply takes a machine in *any* state and configures it.

Areas of Change

Modifications made to UNIX machine file space are often categorized on into the following areas:

- The operating system. The kernel, libraries, server processes, controlling files, initial applications and whatever else the vendor has decided is part of their OS release.
- Software. Any additional software installed on the machine beyond what was installed as part of the OS. In some places, new software installed only goes into /usr/local, while in others, it may go anywhere.
- User space. Home directories, files in /tmp, crontabs, and so on. On UNIX machines, the presence of these files doesn't typically impact the configuration of the machine from an administration perspective.
- Glue. The part of the computer that makes the environment appear like one large system, including such things as hooks for file system mounts. User space and software is often accessible via these mounts rather than residing on local disk.

The distinction between the operating system and the software installed comes to us because of the traditional model in use at many large sites: each individual workstation has its own copy of the OS, while the software is delivered to them from a central server. User space is typically centrally served as well. Thus, the difference between OS and software can often be

"what is on the local machine" versus "what is on the server."

In this distinction, for the purposes of workstation change management, the centrally served software and user space repositories is usually considered to be a different kind of configuration management problem. They are not so much a part of machine configuration as environment configuration. However, software and user data is sometimes installed directly on machines in order to improve performance or reduce the dependency on the network. In these cases, as was seen in the examination of the configuration files at Northeastern, the differences between a configuration distribution scheme and a software distribution scheme can become quite unclear.

This categorization also emphasizes the issue of responsibility for change management, as shown in Figure 5. Interestingly, the systems administrator is involved in each area of change. Note that in the OS and software areas, the sysadmin must work to configure something that was created by someone else. Perhaps that's why systems administration is so hard.

The Role of the Environment Model in Host Changes

The above discussion assumed a model where a machine has its own copy of the operating system, and gets user data and software data from over the network. This is a simple model, useful for understanding changes on a local machine, but very few real world systems conform completely to it.

On one end of the spectrum is the environment where every machine has its own OS, all local software, and all local user space. This is often done because the performance of local disk is so much better than network disk, because the environment is so small that the systems administrator (if there is a designated administrator) hasn't had to discover the value of centralized servers, or simply because everyone at the site is very good at administration of their own machine.

At the other extreme is the diskless workstation or X terminal model, where absolutely no data whatsoever is kept on the individual machines, and everything is served from some set of central locations. This is usually done to make administration easier, but if it's not done right, it can still be quite difficult to manage.

The need for a configuration management system may be less pronounced in some models than in others. Ideally, an abstraction mechanism would be applicable to the entire spectrum of data distribution models. In examining existing systems, it appears that one constant goal is to achieve the reliability and performance of the independent machine model, while achieving the management simplicity and environmental consistency of the diskless model.

Change Magnitude Conjecture

Based on the observations of the orders of magnitude of changes in the Northeastern University configuration files, I propose this model for understanding change magnitude.

Assume an organization with a sufficiently large computing system, and the following sets of files:

- U – the files that contain information that relates to the way in which specific users can use the system
- G – the files that contain information that relates to the way in which a particular group of users can use the system
- E – the files that contain information that defines the services that function in the network and the architecture of the environment
- I – the files that are used to initialize services that then reference centralized information resources

In practical terms, files in set U change more often than files in set G, files in set G change more often than files in E, and files in set E change more often than files in set I. Furthermore, in my observations, the ratio of changes between U and G is approximately the same as the ratio of changes between G and E, and so on.

In psuedomathematical terms, if C(X) means "the number of times that a file of type X is changed," then there is a number k>1 such that

$$C(U) \geq kC(G) \geq k^2C(E) \geq k^3C(I).$$

This postulation has yet to be proven in any formal sense. In order to so, one would, at the very least, have to come up with a more rigorous definition of the sets of files.

However, if it turns out to be generally true then it has important ramifications to those working on configuration management systems. In particular, systems must best support distribution of files with a high change value.

	Initial Responsibility	Configuration Responsibility
OS	Vendor	Sysadmin
Software	3rd party	Sysadmin
User	Sysadmin	User
Glue	Sysadmin	Sysadmin

Figure 5: Responsibilities.

This also points out an interesting problem for systems administrators. In this model, one would expect that files in group U, which are related to specific user information and change the most often, should only impact one user if there were a problem with the change. Likewise, files in group E, which configure system-wide services, should be more likely to impact a large number of users at one time. While this situation is true in general, it is most certainly not true every time. If there is a problem in the NIS passwd file or in the central DNS entries, it is entirely possible for it to take down an entire environment.

We must design our systems so that the changes that are made the most often have the least potential for negative widespread impact.

The State of the Community

There is a disturbing dichotomy in the systems administration community. The experienced administrators with whom I've discussed the contents of this paper generally feel that the area of systems configuration is well understood and that many of the points contained here are nothing new. This may well be true, since this has been an area of exploration for at least ten years.

At the same time, these administrators and nearly every one of subjects of the site survey indicated a strong dissatisfaction with the system they were using. None of the more sophisticated tools developed by the LISA community were being used at any of the sites that I visited. In fact, each of them used a home-grown tool, often layered on top of rdist or NIS. None of the newer administrators were aware of the work that has been done by the community, and may be doomed to putting out fires until they too have developed 20/20 hindsight and specialized scripts.

Even though our environments are changing like mad, our standard methods for handling the changes have remained largely the same. It is to be hoped that a deeper understanding of the area will help to solve this problem.

Towards a Stronger Abstraction

I began this paper by suggesting that systems administration community needs stronger abstraction models in order to manage complexity. Throughout this paper, I have made observations that could be factored into the creation of such an abstraction. I would like to close by summarizing a few key points about possible abstraction models.

A good abstraction model changes the way in which one thinks. It presents an interface and hides implementation details. One should be able to think in terms of "updating the environment" rather than in terms of pushing changes out to hosts.

It may be necessary to change the configuration model in order for it to support strong abstraction. A few ways to do this were suggested earlier:

- Migrate changes into the network and away from the host. The aliases file at Northeastern is a good example of this model. Rather than make changes on every host's aliases file, or even make changes to the NIS aliases map and push them out, all changes are made on and isolated to the primary mail server. The passwd map under NIS is a bad example of this. Even though the passwd map is the source of the vast majority of changes, the local /etc/passwd file must still be updated regularly on client machines in order to change root passwords or update the netgroups in the file.
- The usual model is "configure from," i.e., one assumes certain information about a host (for example, that it is up) and makes changes to that configuration to get to the desired state. An alternate model is "configure to," where one simply describes the desired final state, and it is up to the machine to figure out how to get there. The MIT Athena project [Rosenstein] used a version of this model rather extensively, but it seems not to have caught on much outside of the Athena environment.

Furthermore, it should be possible to instrument and evaluate any methods or tools being used to implement the abstraction. Libraries are often available with debugging and profiling information to allow programmers to improve the quality of the code that calls the library routines. One can compare library routines and see which performs better, even if one doesn't know the details of the code. We need to be able to measure our tools and understand whether or not they have improved the quality of our systems administration. This may require some kind of analysis tools or formal models of the abstraction, perhaps allowing one to describe the environment and the changes applied to it in some kind of state diagram.

The ultimate goal is to improve systems administration by making it easier to manage large and complex systems. Hopefully, this study of configuration mechanisms in practice today will help the systems administration community move one step closer to that goal.

Acknowledgements

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

My sincere thanks goes out David Blank-Edelman, Michele Evard, Bill Nickless, Gail Pieper, Gene Rackow, and members of the MCS Support Group for moral support and suggestions for this paper. In addition, I would like to express my gratitude to the many anonymous folks who participated in the site case study.

Author Information

Rémy Evard is the Manager of Advanced Computing Technologies in the Mathematics and Computer Science Division of Argonne National Laboratory. Among other things, this means that he looks back fondly on the days when he had time to crash the system in spectacular ways while exploring weird administration ideas. He can be reached at evard@mcs.anl.gov.

References

[Anderson] Anderson, Paul, "Towards a High-Level Machine Configuration System," *LISA VIII Proceedings*, 1994.

[Fisk] Fisk, Michael, "Automating the Administration of Heterogeneous LANs," *LISA X Proceedings*, 1996.

[Harlander] Harlander, Dr. Magnus, "Central System Administration in a Heterogeneous Unix Environment: GeNUAdmin," *LISA VII Proceedings*, 1994.

[Imazu] Imazu Hideyo, "OMNICONF – Making OS Upgrades and Disk Crash Recover Easier," *LISA VIII Proceedings*, 1994.

[JumpStart] Sun Microsystems, http://www.sun.com/smcc/solaris-migration/tools/docs/cookbook/30.htm.

[NIS] Sun Microsystems Inc., "The Networking Information Service," *System and Network Administration*, 1990.

[Rosenstein] Rosenstein, Mark A., and Geer, Daniel E., and Levine, Peter J., "The Athena Service Management System," *Proceedings of 1988 USENIX Conference*, 1988.

[Rouillard] Rouillard, John P. and Martin, Richard B., "Config: A Mechanism for Installing and Tracking System Configurations," *LISA VIII Proceedings*, 1994.

[Shaddock] Shaddock, Michael E. and Mitchell, Michael C. and Harrison, Helen E., "How to Upgrade 1500 Workstations on Saturday, and Still Have Time to Mow the Yard on Sunday," *LISA IX Proceedings*, 1995.

[Stern] Stern, Hal, "Managing NFS and NIS," O'Reilly and Associates Inc., 1991.

[Stone] Stone, Ken, "System Cloning at hp-sdd," *LISA I Proceedings*, 1987.

[Tivoli] Tivoli Systems, *Tivoli Management Environment*, 1992.

[Zwicky] Zwicky, Elizabeth D., "Typecast: Beyond Cloned Hosts," *LISA VI Proceedings,* 1992.

Tuning Sendmail for Large Mailing Lists

Rob Kolstad – Berkeley Software Design, Inc.

ABSTRACT

One of BSDI's mail servers hosts what might be the Internet's busiest mailing lists: inet-access@earth.com. This list now has about 2,000 subscribers and occasionally processes traffic as high as 200 separate messages per day. This 400,000 message per day aggregate is a taxing load for a non-optimized mailing system.

When the project started, the mail queues sometimes lagged as much as five days (hundreds of thousands of messages) behind. A single message could take more than five hours to attempt delivery to members of the list. Furthermore, the disk load was high as sendmail processed the queues repeatedly (trying to deliver previously missed mail). All in all, the system's efficiency was remarkably low.

This paper describes the procedures undertaken to reduce delivery times to under five minutes (for all 2,000 subscribers) and mitigate problems associated with unavailable hosts.

Outline

First, the paper discusses the problem of large mailing lists and processing them. After outlining the goals for the new processing system, the state of the old system is described. The methodologies employed to reduce overhead and increase throughput are then discussed along with tools for measuring and aiding performance. Finally, the final state of the system is presented.

The Problem

Mailing Lists

Mailing lists are surely the first (and arguably only) 'push technology.' New information is moved toward the consumer shortly after it is produced. These with online mail systems (vs. POP-style mail systems) who additionally employ *biff*(1), can receive notification of mail delivery in real-time. This convenience and timeliness have motivated mailing lists' popularity since the first days of ARPANET mailers. As the 'net proliferates, lists are increasingly for a variety of discussions and for a variety of communities.

Mailing list sizes run from the tiny (a handful of recipients) to the huge. InfoBeat (formerly Mercury Mail) creates customized content for various mailing lists and (in August, 1997) sends out more than 200,000 messages every afternoon.

Surprisingly, mailing lists continue to be popular in spite of other technologies like USENET News and the World Wide Web. This might be because of the pure 'push' properties of the mailing lists. I hypothesize that people have integrated mail reading into their daily (or hour or minutely) task schedule and that the mingling of mailing lists into such a paradigm is just too convenient to change.

Mailing List Servers

The convenience and efficacy of receiving mailing lists depends on the mailing list servers (hereafter 'list servers' or just 'servers'). If a list server fails to deliver a message in a timely manner (or at all!), much of the effectiveness of mailing lists is lost. As lists grow, delivery becomes an ever more resource-intensive procedure that is complicated by all sorts of factors, including: host outages, network link outages, DNS service problems, slow clients, slow networks, and packet loss. Any of these factors can increase mail delivery time from the 50-1000 millisecond timeframe to the multi-day timeframe (as much as seven orders of magnitude).

Furthermore, queueing a large number of messages to a list server can seriously impact its performance on other tasks. Sendmail's standard paradigm for processing disk queues can cause severe resource saturation when mail queues have more than one or two thousand messages ready to be delivered. This saturation can impact server efficiency so that only a few messages per minute are delivered. Tuning such systems is painful since disk operation (including paging in editors, for example) is impacted. Mixing high priority mail delivery (e.g., corporate mail) with a high volume mailing list can produce disastrous and unacceptable results like 12 hour delivery times for mail from one desktop to another desktop in the same office.

The properties suggest a set of goals for a good mail server environment.

Goals for Mailing List Servers

The combined needs of the message recipients and list administrators yield a set of common-sense goals:

- Fast delivery of messages (low latency)
- Reasonable consumption of server resources

- Easy (or least low time commitment) administration
- Use of existing tools for list processing
- Ability to monitor results to ensure goals are being met

Delivery speed is surely a primary consideration of mailing lists. Arguments can be made that reasonable maximum delivery times run anywhere from a few minutes to an hour. Analyses of "as fast as possible" suggest that a maximum delivery time of 5-10 minutes for a few thousand users is reasonable and acceptable to the vast majority of users.

Reasonable server resource usage is an important goal for list servers. Otherwise, they do not scale well with increased list size and become too expensive for organizations to support (both in terms of hardware resources and both direct and indirect administration costs).

Reasonable administration costs is another important goal. List servers that require more than a few minutes of daily maintenance start to become problematical when:

- the list administrator is unavailable (e.g., for vacation or a conference),
- management is looking for ways to reduce overhead, or
- the personal patience of the administrator is tried (for any reason).

Use of existing tools is a prerequisite for continuing goals of low-impact administration, low-impact costs (both direct and indirect) for running a mailing list, and ability to share innovations with other list administrators. Our site would probably opt out of running mailing lists if costs for extra software were required.

Monitoring performance and resource usage becomes important once the hardware costs of server mailing lists are exposed. Neither network bandwidth, server hardware, nor administration time is free. If high quality service is not in the offing, these costs are best directed towards other, higher impact and more effective services.

Initial State

BSDI hosts the inet-access@earth.com mailing list. It currently support approximately 2,000 subscribers with anywhere from 100 to 200 messages per day. At the time this analysis began, the message had just over half that many subscribers and slightly less traffic.

Probably the biggest problem was that delivery times were starting to exceed five hours and the system load seemed to be increasing very quickly. Paging was high; disk I/O was high; machine performance was sagging.

A quick check revealed 50,000 messages in the mail queue. This was surely one of the causes of the high disk I/O as sendmail explored the queue with each new process instantiated to mop up previously undelivered mail.

Checking *ps*(1), showed over 100 sendmail processes running. Normally, this wouldn't seem very painful for a mail server, but in this case each process consumed over 2.5 MB of memory! This caused the paging and exacerbated the disk load.

The load average ran from 5 to 20 even though the CPU was not saturated. The (mostly) busy disks and paging were keeping jobs from running.

Interestingly enough, most queued deliveries would 'catch up' overnight and the mail queues would be quite tolerably small by the time the first morning batch of list mail began to be distributed. This put a certain kind of cap (18 hours) on mail delivery time, but the overall big picture was quite alarming since other mail going through the mail delivery machine would be delivered in hours instead of milliseconds.

The 'Fast Fix'

As a quick-and-dirty fix to increase throughput, Tony Sanders (inet-access's owner and list maintainer) gathered statistics about mail delivery times. Here is a typical line (displayed here broken down into fields) that he used to analyze the delivery time:

```
Sep  6 10:54:25 - date message was delivered
ace - hostname upon which sendmail is running
sendmail[24338]: - process name and PID
KAA24336: - date message was delivered
to=peter.j.scott@jpl.nasa.gov, -
            recipient
ctladdr=kolstad (101/0), - sender
delay=00:00:07, - delay from time message
            was queued until delivery
xdelay=00:00:06, - time for this particular
            delivery attempt
mailer=smtp, - mailer used
relay=mailhub.jpl.nasa.gov.
            [137.78.18.34], -
            destination machine
stat=Sent (Message received and
            queued) - final status
```

On a daily basis, Tony Sanders gleaned the xdelay information for each recipient (across dozens of mail messages in a given day) and then sorted outgoing mail list so that those with lower xdelay averages were delivered before those with higher averages. This had the property of rewarding 'good citizens' with outstanding delivery time (i.e., the first 100 or so good citizens received their mail within a minute of its arrival at the list server). Furthermore, 'better' citizens were not punished by having their mail delayed by multi-minute timeouts behind unavailable hosts. Those people at the end of the queue still waited over five hours for their mail to be delivered.

The First Suggestion

I entered the scene because one of my mailing lists (the USA Computing Olympiad list) was being delivered ever more slowly. What used to take 20 minutes was taking hours.

I checked out the system and observed the symptoms reported above. I suggested that the 1,500 person mailing list be split into 75 lists of 20 recipients. My reasoning was that 75 lists of recipients would be processed in parallel and all the 'waiting' (for hosts to answer, etc.) would be parallelized and there would 'always' be some host ready to communicate. And, of course, how long could 20 deliveries take? I was hoping to deliver all the messages in a minute or two by this increase in parallelism.

In a bizarrely political process, I was completely overruled by our system administration staff who complained quickly and bitterly that we could not possibly support the RAM and process slot requirements of 75 parallel sendmail processes. The fears were based on the notion that three or four messages would arrive in a small interval and thus stress the system with 4 x 75 = 300 processes and, worse, 4 x 75 x 2.5 = 750 MB of virtual memory requests. I was surprised that sendmail used up so much data space (since the code space is shared), but I am a strong believer in delegating authority and this particular authority had in fact been deleted.

Second Try, First Suggestion

I bargained with the administrators. I suggested that we split list into four lists of 375 recipients each. They were not pleased. However, being the company president, the subtle force of that office won the day. We fixed the outgoing mailing list processor to mail to four different aliases of 375 recipients instead of one big alias of 1,400 recipients.

Primitive analysis tools showed a an improvement in throughput of roughly 4x. 'Maximum' mail delays were reduced from over five hours to less than two hours. The delivery rate was increased proportionally. The number of processes in use did, in fact, increase. The RAM usage was high but not nearly as high as the 2.5 MB per process that had been feared.

It was difficult to measure throughput since the mail queue status indicators are only updated every ten message deliveries. It was at this point that we committed a huge error and changed the update rate in the /etc/sendmail.cf file from 10 to 1:

```
# checkpoint queue runs after every
# N successful deliveries
O CheckpointInterval=1
```

Do not do this at home.

An Aside on the Environment

For better or for worse, the inet-access mailing list environment was a 'live' environment. Everyone involved in this project had to take care not to endanger the ultimate throughput of the list. We knew that we lagged as many as 50,000 messages by the time prime-time for mailing ended each day. A bad move would result in getting more than one day behind and, thus, potentially failing to catch up overnight when the traffic was reduced. We feared that falling too far behind would leave us unable to catch up ever.

Second Experiment

Heartily encouraged by the results of four lists instead of one, I wanted to increase the number of lists while decreasing the number of users per list.

It was clear at this point that monitoring tools were necessary that could:

- monitor the instantaneous rate of delivery and
- summarize the day's performance.

Without such tools, evaluating the status of a new experiment would be difficult if not impossible.

It was decided to watch processes, RAM use, disk I/O rates, and network I/O rates. A bottleneck in any of these areas would cause a 'hard limit' on performance.

As the number of recipients in each 'list chunk' was decreased, delivery rates and throughput increased.

This was puzzling, to an extent, because of the initial fears of process table crowding and virtual memory consumption. Observation of the RAM use showed that a sendmail process delivering a single message never used much RAM. Later observations showed that only 'older' sendmail processes used lots of RAM, suggesting speculation about a memory leak or a table that grows with status information about destination hosts. At any rate, having solved the puzzle, it was easy to envision ever more list chunks with ever smaller sets of recipients.

Disk I/O continued to increase as the number of entries in /var/spool/mqueue increased. Network I/O never got very high. Our T-1 line was never seeing much usage, never more than 10-15%.

Delivery times had declined to less than one hour. This caused queue sizes to be considerably reduced so that list delivery was keeping up with the lowest possible expectation of throughput. This was a good milestone.

Head Scratching

We observed RAM usage was now relatively low. We were trying to find the bottlenecks that were reducing throughput. We observed:

- Disks are getting busier
- CPU isn't that busy
- Load average isn't getting worse

- Network isn't the problem

The actual cause of less-than-maximum throughput was unknown.

Maximum Throughput

Of course, when one is concerned about achieving maximum possible throughput, one should try to figure out precisely what the maximum throughput could possibly be.

I wrote a small program, 'mailtest.c.' This program implemented what is arguably the smallest set of steps necessary to deliver a message, it:

- Opened port 25 on the destination machine (note that DNS resolution is in some other process),
- ran the SMTP protocol,
- sent approximately the shortest message that the mailing list saw,
- closed the remote port, and
- exited.

On a PentiumPro/200 processor, this task (in repeated observations) ran in about 30 ms. This is implies that the maximum throughput of a mailer (under the most ideal of all possible conditions) running on a PPro200 processor with a nearby recipient machine whose name was already resolved was:

3600 sec/hour / 30 ms/message = 120,000 messages/hour.

The main bottleneck for this particular experiment was the CPU though the network was running close behind.

Statistics

It was time to build tools. The first tool ('batchstat') was designed to show a day's summary of mail throughput. See Figure 1 for a sample of its output.

Initial observations were performed by watching the output of the mailq command. Tony Sanders wrote a slightly modified version called mailqq that showed the number of messages to be delivered instead of the names of all recipients for a message. It also found the sum for all messages waiting to be delivered.

The process of running mailqq while the /var/spool/mqueue queueing file update interval was set to update after each message was delivered gave a relatively depiction of mail throughput, though its effect on disk performance was quite debilitating (since it read so many files to find its statistics). More on this later.

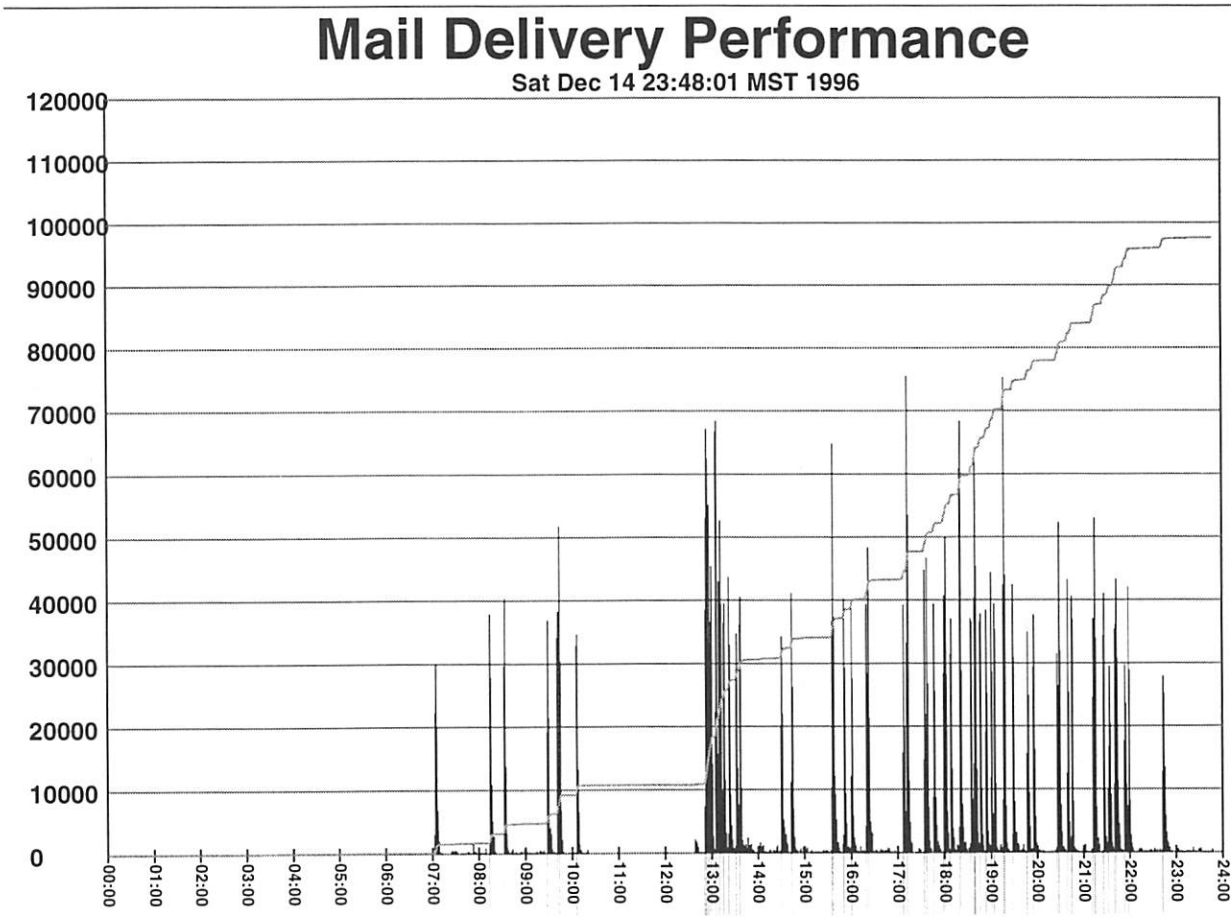

Figure 1: Initial batchstat output.

batchstat

The batchstat program labels the output page, the left axis in a processing rate measured in messages/hour, and the bottom axis in time throughout a single day. Note that statistics for a day often start at 2:00 am instead of midnight. This doesn't change our observations, though.

The dark black vertical spikes running up from the X axis depict 'instantaneous message delivery rates' through the day. These are actually averages over a minute or two. The light vertical lines below the X axis show when a new message arrived for delivery to the mailing list. Later versions of this program went out of their way to ensure at least one pixel between these bars so that high arrival rates could be observed accurately.

The lighter gray bar that slowly increases in value across time is the integral of the spikes: it shows the total messages delivered so far since midnight. In this example, about 98,000 messages were delivered across the day.

The obvious goal of a mailing list delivery program is to deliver all the messages the instant a message arrives. On this graph, such behavior would display as a spike whose height is infinitely high for a very short period of time. Higher thinner/narrower spikes for each incoming message show better performance of a mailing list handler.

realstat

Waiting an entire day to see if the spikes showed up for mail deliveries was a nailbiting experience. The realstat program read the realtime output shown in /var/log/maillog so it could display performance in a small window. Figure 2 shows realstat's output during a slow period.

```
 9:00:00   480/   2
 9:00:15   960/   4
 9:00:30   960/   4 --
 9:00:45  2160/   9 *--
 9:01:00  1920/   8 *--
 9:01:15   720/   3 --
 9:01:30  2400/  10 **--
 9:01:45   720/   3 --
 9:02:00  1440/   6 *-
 9:02:15   960/   4 -
 9:02:30   960/   4 --
 9:02:45  1440/   6 *--
```

Figure 2: Realstat Output During Slow Period

The first column shows the time of the observation. The observation time is actually the 15 seconds leading up to the time of observation. The second column shows the hourly rate of message delivery. The next column shows the actual number of messages delivered in the previous interval. This number is scaled and presented graphically as '*'s. The set of dashes shows, in the same scaling, the number of messages that were attempted to be delivered but, for some reason, failed. For this graph, far more messages failed to be delivered than succeeded.

mailstat

The mailstat program calculates a numerical summary of messages delivered since the /var/log/maillog file began. See Figure 3 for sample output.

```
mailstat: Sat Feb 22 09:05:40 MST 1997
                 failed          deliveries
MMM DD HH mhosts/recipt     mhosts/recipt
=== == == ============== ==============
Feb 22 02     433/   612    1111/   1111
Feb 22 03     495/   696    1298/   1298
Feb 22 04     431/   615    1137/   1137
Feb 22 05     421/   610     810/    810
Feb 22 06     422/   606     717/    717
Feb 22 07     411/   587     931/    931
Feb 22 08     427/   616    1039/   1039
Feb 22 09      22/    22     105/    105
========= ============== ==============
Totals       3062/  4364    7148/   7148
```

Figure 3: Mailstat output, slow day.

The first three columns show the date and hour for the summary shown to the right. Subsequent columns show the number of hosts that a message to which a message actually failed to be delivered and the number of messages that failed (which, surely, is always at least as high as the number of hosts). Similar statics follow for successful deliveries. Figure 4 shows mailstat output when the queue is full of messages that can't be delivered for some reason.

```
mailstat: Sat Feb 22 09:03:49 MST 1997
                 failed          deliveries
MMM DD HH mhosts/recipt     mhosts/recipt
=== == == ============== ==============
Feb 22 03   18811/ 18811     522/    522
Feb 22 04   27065/ 27065     574/    574
Feb 22 05   29342/ 29346    1738/   1887
Feb 22 06   29973/ 29978       8/      8
Feb 22 07   26668/ 26675    1556/   1690
Feb 22 08   11768/ 11787    1347/   1464
Feb 22 09    1893/  1896     566/    606
========= ============== ==============
Totals     145520/145558    6311/   6751
```

Figure 4: Mailstat (few deliveries, many failures).

Further Experiments

We continued to increase the delivery parallelism. Eventually, there 100 lists of 15-20 people each. This caused ever-decreased delivery time, a very busy machine, and incredibly busy disks.

Examination of the disk I/O rates showed that each mail delivery was causing lots of disk I/O. This brought to mind the update of the queueing files for each mail delivery. Because sendmail is super-safe in its algorithms for changing on-disk data structures,

many of the operations for updating the queueing files ended up requiring synchronous disk operations. While this enabled observation of disk queue sizes, it required an unacceptable disk I/O load burden. Besides, one can observe /var/log/maillog to get an even better understanding of statistics for mail delivery. There was no double that synchronous disk operations were destroying performance.

The configuration file was changed to 'update the queueing files every 10 deliveries.' This change immediately removed the disk I/O bottleneck from the system's performance. Figure 5 shows 275,000 messages transmitted in one day with very high throughput. Even though the throughput is averaged across 60 seconds, some of the spikes are trending toward the theoretical maximum throughput (though the conditions are far less than ideal).

Next Analysis Step

Trying to characterize both the high and low performance periods led to a certain set of discoveries. First, the number of hosts unavailable for delivery has a profound impact on throughput. Why? Because a sendmail process waits a long time in the vain hope that a response might be received to the open on port 25. Second, the number of messages available to deliver impacts performance. How? Because when lots of messages can be delivered, more processes can run in parallel and deliver more messages per unit time.

It was observed that the mail queue still had substantial size, even though messages were being delivered quickly through the day. Figure 6 shows some (edited) mailqq output when the queue size was nonzero.

Each message has a few 'stragglers' that are not being delivered. A quick perl script ('latedudes') shows the message counts and names of recipients with messages in the mailqueue:

```
 82 todd@acc.com
127 glennh@netstation.net
127 ispmail@zhi.dialup.access.net
127 jnussbaum@americandata.net
127 kevinc@rrt.com
127 mp3@cyber-gate.com
127 nevin@shadowave.com
127 tcosta@biznm.com
164 rdavis@masschaos.de.convex.com
200 whenpigsfly@worldsrv.net
```

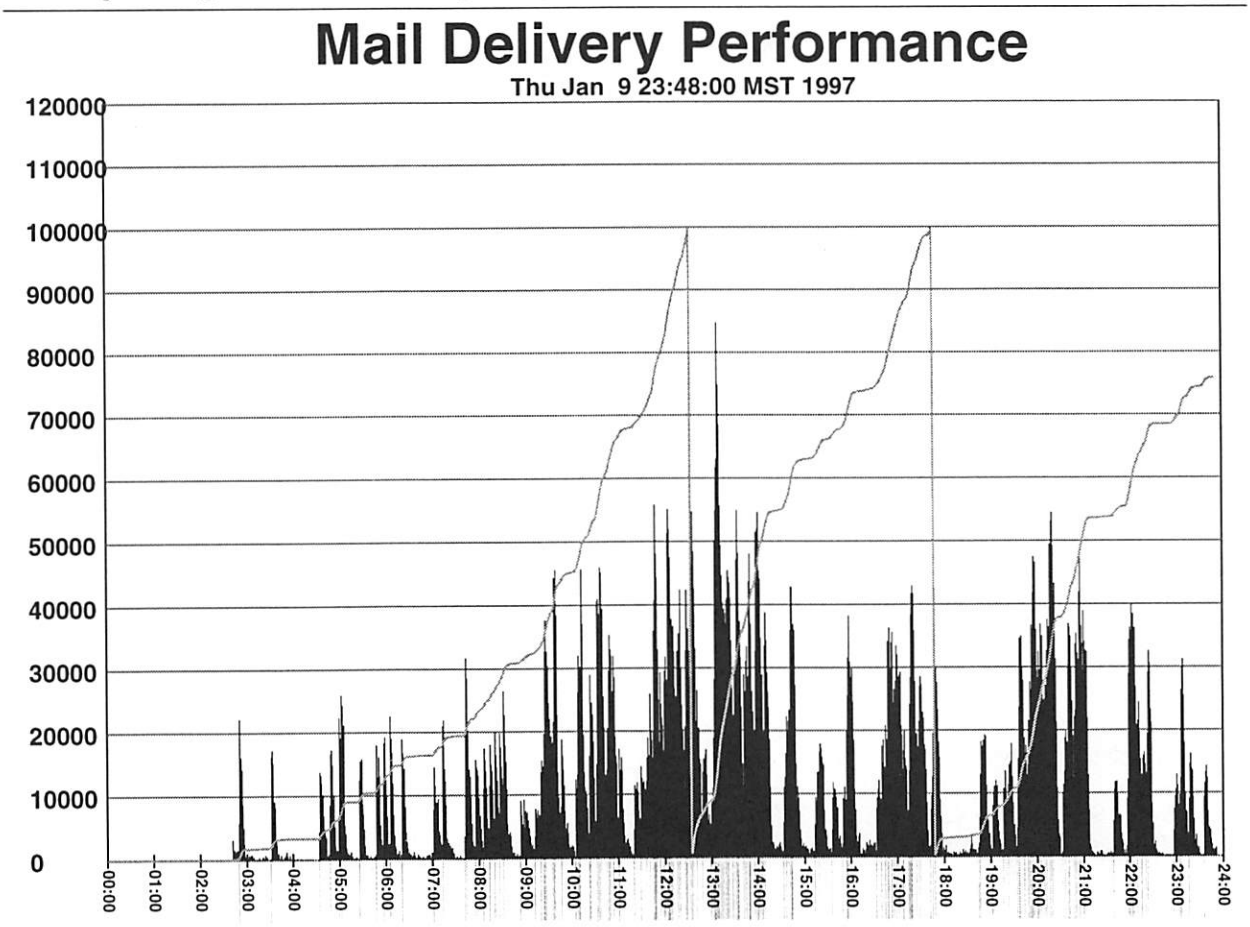

Figure 5: Improved throughput (small message queues, disk bottleneck removed).

```
388 cbrown@matnet.com
559 robert_thompsen_at_usr-css...
593 berney.ortiz@mailserver.oi...
595 list.inet-access@optimum.net
```

What a surprise to find some recipients had not accepted mail for many days. In the worst case, each of these messages would have its own queue file in the mail spool.

The impact of each of these recipients can range from the trivial to the dramatic. Here's what happens when sendmail tries to send to a host that is 'busy, hung, or dead.' First, sendmail tries to connect to the host. Maybe the connection succeeds (but host turns out to be slow or net is losing lots-o-packets). Maybe the connection fails.

Each step in the SMTP protocol from connection through completion has long time-out (like as much as 300 seconds). This means that a particular sendmail process idles for five minutes waiting for a reply. This reduces throughput – especially when 100 sendmail processes are conducting this exercise in parallel.

At any point in time, 1-3% of recipients – and these recipients are ISPs – are unavailable. InfoBeat reports a number closer to 10% for average users. This is not to say that users and ISPs have control on all possible outages – cable cuts do make a difference. Nevertheless, in a mailing list with 1,400 - 2,000 participants, 2% is anywhere from 28 to 40 recipients. This means a total of 6,000 to 8,000 messages per day can not be delivered!

Next Step

Obviously, it would be advantageous to reduce timeouts for initial contact/mail transmission. This would enable 'good citizen' sites to continue their good throughput. Some sort of 'reaper' process could come along later to deliver to slower (or dead) hosts.

Happily, all these times are configurable in sendmail. We reduced them 5x. This sped up initial mail delivery, though some messages were, of course, not delivered. A second sendmail.cf file with slower timeouts was created and run three times/hour.

Note that this is all in the context of sendmail already remembering when a host is unavailable and not trying that host again for a one hour period.

Reducing Queue Search Time

The most painful part of running mailing lists is the *manual* removing and adding people to the list. For the inet-access list, Tony had a policy of not removing people from the list for a bounce or even for two days of list bounces. This made the outgoing queue grow significantly.

So we created 10 more queues to run separately. We moved jobs moved from queue to queue when older than a certain amount of time. We automatically scheduled ever more 'reaper' processes to run those (presumably smaller) queues.

Regrettably, it never seemed to help. Performance differences were barely measurable, if at all. This idea was abandoned.

Summary of Modifications So Far

All in all, it doesn't take many changes to speed sendmail's throughput dramatically.

First of all, use lots of parallelism (hundreds of processes in parallel). Of course, one should reduce impact of unavailable recipients by keeping track of hosts that won't answer and by reducing the timeout for hosts that can't keep up with standard Internet speeds.

Stragglers continue to have effects by slowing mailq commands (which touch each file) and slowing sendmail itself for queue runs (which also touch each file).

The mail delivery time was reduced and throughput increased with ever better spikes; see Figure 7. In fact, the high load performance is also outstanding. Figure 8 shows deliveries after one main spooling machine was down until noon one day.

```
[...]
nrecipients  length  date                  sender
3 BAA08598   1554 Sat Feb 22 01:02 <inet-access@earth.com>
3 BAA08677   1017 Sat Feb 22 01:29 <inet-access@earth.com>
3 FAA10201   1438 Sat Feb 22 05:24 <inet-access@earth.com>
3 FAA10208   1438 Sat Feb 22 05:24 <inet-access@earth.com>
3 HAA10369   1527 Sat Feb 22 07:46 <inet-access@earth.com>
3 IAA10524*   423 Sat Feb 22 08:52 <inet-access@earth.com>
4 HAA10371   1527 Sat Feb 22 07:46 <inet-access@earth.com>
4 HAA10383   1527 Sat Feb 22 07:46 <inet-access@earth.com>
4 IAA10544    423 Sat Feb 22 08:53 <inet-access@earth.com>
4 IAA10558    423 Sat Feb 22 08:53 <inet-access@earth.com>
5 IAA10541*   423 Sat Feb 22 08:53 <inet-access@earth.com>
```

Figure 6: Mail queue analysis after several nondeliveries.

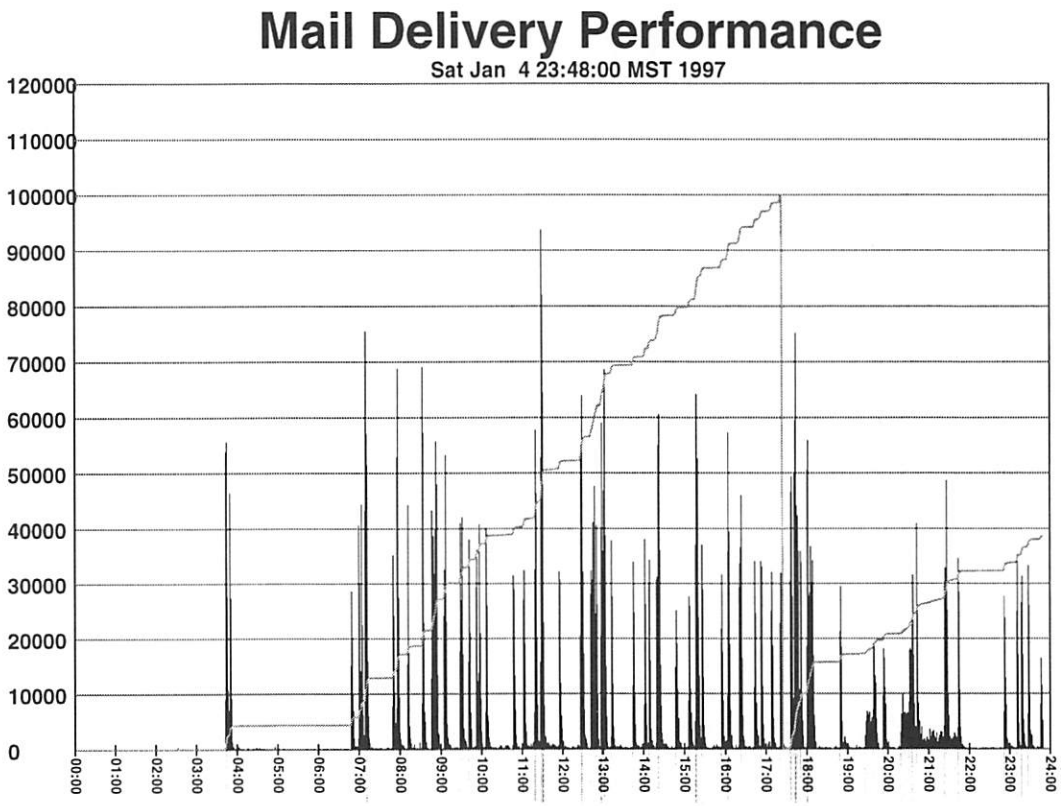

Figure 7: Good performance after modifications.

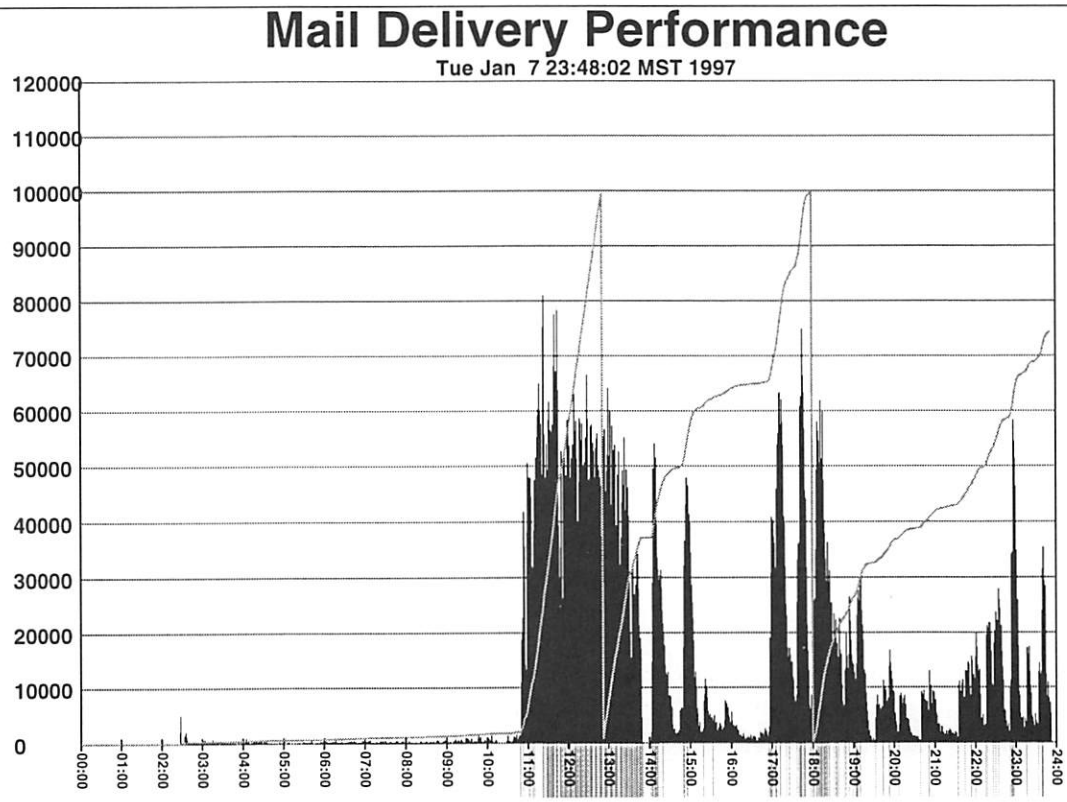

Figure 8: Good performance under load.

Futures

Now that the system is up, running, and manageable again, it's easy to see some of the future improvements that might be tried.

First of all, stragglers should be coalesced into a single message/recipient-list pair. This would reduce the queue sizes trivially. One can even envision creating a large digest for stragglers and having one queued file per straggler instead of one queued file per message. Obviously, it depends on the ratio of stragglers to messages per day.

Secondly, one might consider a policy of just deleting messages older than, say, 24 hours. These people can always look at an archive.

A grandiose plan (and I do mean unmanageable) would be to rewrite sendmail for a high – but constant – number of concurrent transmission processes coupled with the use of extended SMTP to send multiple messages once a machine is up. This would make sendmail treats its queues in a sort of contrapositive way to the way it treats them now (sending all messages to one site rather than one message to all sites).

To be fair, though, sendmail already close to achieving highest possible bandwidth for a given speed of network connections. It's not clear any of these measures would increase actual throughput.

Also note that even with these improvements, messages smaller than 10KB or so don't push T-1 speeds to their limit yet. It takes pictures for that.

Availability

All scripts mentioned here are available by sending a short yet descriptive e-mail request to the author <kolstad@bsdi.com>.

Conclusion

It is not that difficult to reduce mailing list latency dramatically. We now have a script to insert in /etc/aliases to break message into parts. The time to deliver 95% of a mail queue was reduced from 5 hours to 3.5 minutes. The unavailability of recipients is still the biggest performance problem.

And, most amazingly, sendmail does about as good as can be done in delivering large amounts of mail! Figure 9 shows the latest performance figures as traffic has increased and a new file system design has reduced file I/O dramatically.

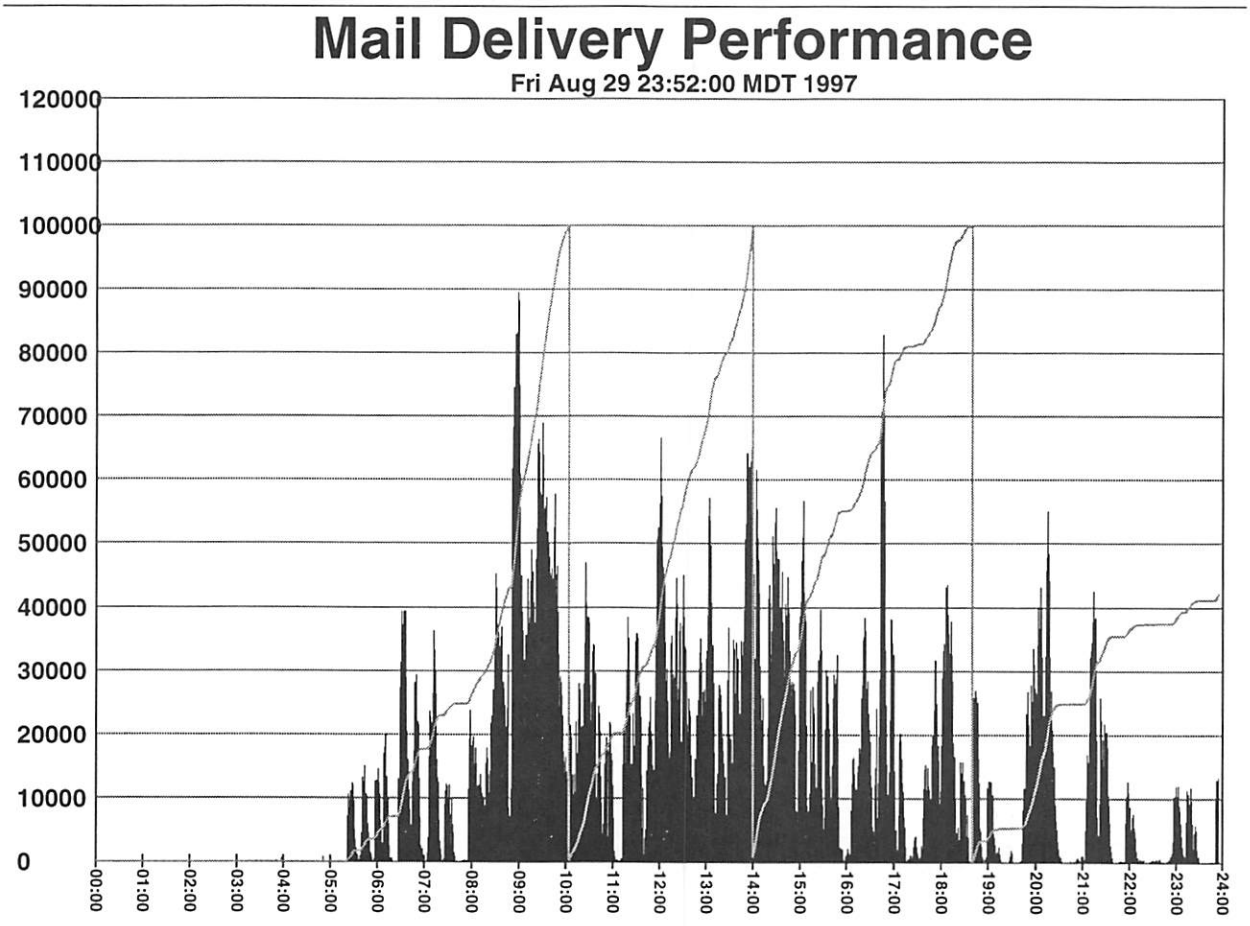

Figure 9: High traffic with new file system.

Selectively Rejecting SPAM Using Sendmail

Robert Harker – Harker Systems

ABSTRACT

With the growing popularity of the Internet, unsolicited electronic mail (spam) has become a major concern. It fills up user's mailboxes, clogs mail relays, wastes postmaster time, and creates ill will for sites that have been used as a relay. Most sites want to filter spam before they receive it but filtering spam is hard to do without filtering legitimate mail messages.

This paper discusses what characterizes spam and describes rulesets that can be added to a *sendmail* version 8.8 [1] *sendmail.cf* file to selectively reject mail from specific addresses, domains or IP addresses and to prevent spammers from relaying mail through a site. It discusses the different issues facing corporate sites and the special issues facing Internet Service Providers (ISP). The rulesets presented have been implemented as *M4* template files so they can be easily integrated into a sendmail 8.8 sendmail.cf file as a FEATURE using M4. These rulesets are currently in use at Harker Systems and other sites, and are available via anonymous ftp.

Motivation

Unsolicited email, or spam, has become a chronic problem on the Internet. It fills user's mailboxes and clogs SMTP mail relays. It creates ill will and wastes postmasters' time. Filtering spam is a ongoing challenge.

It is easy enough to filter out spam which has valid addresses from a known spam site. Regrettably, spammers are very creative in how they disguise their messages. Also, one person's spam is another person's interesting junk email. And, if a non-spam message is sent from an address which is defined to be a spam address, it will be rejected.

Because of these issues, filtering spam must be handled with care. It is as much a policy and procedure problem as it is a technical problem. Creating the tools to filter spam is relatively easy. Creating the policy and procedure of what constitutes spam at your site, and what to block, is much more problematic.

One of the important new features of sendmail version 8.8, is a new set of four rulesets which are used to reject mail messages. The check_, *check_relay, check_mail, and check_rcpt set of rulesets allow rejection of a message during SMTP reception – before it is transferred to the destination or a relay. This off-loads the processing and generation of a bounce message by the mail host or SMTP relay. An additional ruleset, check_compat, can used to reject mail during sendmail's delivery of a message after it has been accepted.

Implementation of these rulesets is evolving with versions developed by Eric Allman [2], Claus Assmann [3]. The results presented in this paper represent a next step in this evolution. My students' questions caused me to think about what characterized spam and to add additional tests to the rulesets. The spammers courteously provided copious quantities of sample spam messages so I could test my assumptions.

What Is Spam?

The simple answer is that spam is any unsolicited email that is sent to a large list of users without their prior permission. When someone receives a piece of spam, they recognize it as such and delete it.

Human recipients use their experience and the context of the message to make a decision about whether or not the message is spam. They use various criteria for deciding that a message is spam such as:

- The sender's address or domain name.
- The subject of the message.
- The text of the message.

Unfortunately, in sendmail, an SMTP relay has limited access to the information in a message without significantly modifying the sendmail source code itself. During the reception of an SMTP message sendmail has access to:

- The sender's envelope return address passed in the MAIL From: command.
- The recipient's envelope address passed in the RCPT To: command.
- The host name the SMTP client announces in the HELO or EHLO command.
- The client's IP address and port number from the socket structure.

Information about the client derived from the above information and DNS.

Once the message is accepted for delivery, sendmail has access to additional information, but it is of either limited use, or impossible to use without modifying the sendmail source code. Because of this, using sendmail to filter spam is best limited to the information available during the reception of an SMTP message.

A major problem with rejecting spam using arbitrary filtering criteria is that desirable messages may be rejected along with the unwanted spam. This may range from an inconvenience for a user, to the loss of business due to rejecting a potential sales lead or business opportunity, to an extreme of a legal lawsuit charging violation of the spammer's constitutional free speech rights.

Criteria For Accepting Mail

The criteria used to accept local sender addresses with the rulesets presented in this paper are:

- Local host names
- Local domain name or local virtual domain names

Mail being sent either to or from this host or domain should be accepted. For a standalone mail host this is simple: Does the sender or recipient address contain my host name? A corporate SMTP relay is also simple: Does the sender or the recipient address's host names end with our domain? For a company with multiple domains or an ISP the test is slightly more complex because there are multiple domains that should be accepted as local. You must also make sure the local addresses are truly local and not non-local local addresses such as:

```
user%spam.dom@my.dom
```

Criteria For Rejecting Mail

The criteria that can be used to reject spam mail are:

- Host and domain names of known spam sites
- Bogus sender or host domain
- Bogus sender address
- Bogus user address
- IP networks and addresses of known spam hosts
- Invalid DNS host name
- Spam addresses hidden behind a local name

Some of these are good, but others run the risk of bouncing valid messages.

Known Spam Sites

The most straight-forward criterion for rejecting mail is to reject mail from known spam sites. This is simple to do. Make a list of user addresses, host names or domain names that you deem to be spam only sites or addresses. When the sender address or sending host is one of these spam addresses, it is rejected. While straight-forward, this is the most easily and frequently circumvented filter. Many spam sites do not give valid sender addresses to avoid getting bounce messages to bad recipient addresses, and to avoid getting spam or flame mail in return.

Bogus Sender or Host Domain

The next obvious criterion to filter on is bogus sender, host or domain name. When an invalid sender address is given, the host or domain name are frequently not in DNS. Rejecting mail from hosts and domains that can not be looked up in DNS will get rid of this class of mail. But simply looking up a host or domain name in DNS may cause a site to reject valid mail since many sites have incomplete DNS information. If a sender's host name is not registered in DNS, this criterion will cause mail from this sender to be bounced. A site can choose that this is an acceptable rejection.

A more careful approach is to search the host name and check if the sub-domain or parent domain is valid. If a domain match is found, the host name should be treated as a valid host name.

Bogus User Address

A desirable criterion for rejecting mail is to filter on bogus user address. However, testing for a bad user address is much harder because, short of sending a message to that user address, there is no reliable way to check the validity of the address. A simplistic test for a bad user address might be to connect to the sender's SMTP server and use either the SMTP VRFY or RCPT command to check the address. If the server does local delivery of the message then this would work well.

However, the reality is that most of the SMTP servers are simply SMTP relays that forward the message to an internal host. These hosts will usually accept mail for any address that ends with the local domain(s).

Next you might test the next host to which the message would be relayed. Unfortunately, most SMTP relays either do not return information about to which host the mail will be forwarded, or do not allow connections to that next host.

The other issue with using this criterion is that sendmail does not have any built-in feature that would allow the sender's complete address to be tested. Rejecting bad host or domain names only is an accepted limitation of this paper.

IP Networks and Addresses of Known Spam Hosts

The next set of criteria are based on information about the SMTP client that is connecting to the SMTP server. When a client connects to sendmail, sendmail receives information about the client:

- The host name given in the SMTP HELO or EHLO command
- The IP address of the client from the socket structure
- The TCP port number of the client from the socket structure

The first test is to check the host name given in the SMTP HELO or EHLO command as well as the DNS name of the SMTP client against the list of known spam sites. This is done by running the client host name against the same set of tests that you used for the sender's envelope address.

The second test is to check the IP address to see if it is a known spammer IP address or network. To do this, a list of known spammer IP address and networks

is created. If the IP address matches one of these IP addresses or is a host on one of the known spammer IP network address ranges then the message is rejected.

Rejecting on specific IP addresses is reasonably safe. If a spam site is sending mail from their host, all mail from that host is probably spam. Rejecting mail based on an IP network is more risky because the spammer may be using an IP address allocated to an independent ISP. If you reject mail based on the ISP's IP network number, then you will not only reject mail from the spammer, but also all of the other valid users of that ISP.

For example, if you rejected mail from aol.com's or netcom.com's networks you would block a lot of spam mail, but you would also block a lot more valid email.

Three additional tests are based on the PTR records for the client's IP address. First do a reverse lookup of the client's IP address to check for a valid PTR record. If no PTR record is found then reject it as an invalid or unauthenticated IP address. Next test if the PTR record matches the client host name given on the SMTP HELO or EHLO command. Third, do a double reverse lookup of the IP address to check that the IP address used is actually the IP address for the host name returned by the PTR record.

All of three of these tests risk bouncing more valid messages than spam messages. For various reasons a significant percentage of SMTP clients either do not have PTR records or the PTR record does not agree with the client SMTP host name or with the A records for that host. This could be because of poor DNS administration or because SMTP clients announce themselves as the domain.

However, where these tests might make sense is for messages from specific hosts or domains where you either control the domain, or where your relationship with the domain administrator would permit incorrect DNS information to be corrected. For example, if acme.com and abc.com have a business relationship, it might make sense to check that mail with a sender address of acme.com actually has consistent DNS information. If the SMTP client host name or IP address do not agree for these networks then reject the message. This test is more applicable to rejecting forged email, not rejecting spam.

Invalid DNS Host Name

Test for invalid host names by canonicalizing the address. A trailing dot is added to valid hostnames when sendmail 8 looks it up in DNS with $[hostname$] in the rulesets. Testing for this trailing dot will reject invalid names. Unfortunately, not all hostnames are registered in DNS. By treating the domain portion of the hostname as a search path, the host's sub-domain, or parent domain can be used to validate the hostname.

Spam Addresses Hidden Behind a Local Name

One technique spammers use is to hide their domain behind a local domain:

```
user%spam.dom@abc.com
```

These addresses can be rejected by borrowing the rules in ruleset S0. These rules check that an address truly is local. Extend them to test for local domains as well as domains the relay treats as local, such as the masquerade domains.

Protecting an SMTP Relay

Many sites that have implemented and paid for robust SMTP relays with high bandwidth Internet connections are being used as spam relays. A spam site will open a single connection to the relay and send a single copy of the message with a list of hundreds or thousands of recipients. The SMTP relay then delivers the messages to the individual recipients.

This causes technical problems including congestion of the SMTP relay queues, impacting the CPU performance and throughput of the SMTP relay, saturation of the Internet link, and generation of excessive undeliverable bounce messages. It causes staff problems, including time spent by postmasters and postmistresses dealing with the bounce messages, and the complaints sent to the owners of the SMTP relay from the spam's recipients. It damages the company's reputation and causes loss of good will when outside people mistakenly assume that the company encourages spam.

Criteria For Refusing To Relay SMTP Mail

Spam has many criteria that sendmail can use to reject relay spam mail:

- Is the sender or recipient a user in the local domain?
- If the sender is a user in the domain, is the client a connection from an IP address inside the domain?
- Are the recipients truly local addresses?
- Are the recipient addresses valid addresses?

Some of these criteria are good; others must be rechecked to make sure they are valid messages.

The major criteria of whether a message is valid, and should be relayed, or is spam mail that should be rejected, is if the sender and/or the recipient are part of the SMTP relay's domain(s).

The simplest case is to check that the recipient address is a local recipient. Mail going to a user within the domain should be accepted because they are part of the domain. The same criteria for accepting a sender as a local address are applied to the recipient address.

Checking the sender address as a local address to decide if the message should be relayed must be done carefully. If the sender address is a user inside the domain, and the message is being relayed either to the

outside world, or to another user within the domain, then the message should be forwarded. You should not blindly trust the sender's address because the spammer can simply forge a local sender address. Further checking is called for.

See if the client's IP address originates within the domain. If the client's IP address is not part of the domain, you can assume that the sender's address has been forged. (Restricting the sender address on the basis of the sender's IP address also has the side benefit of reducing or eliminating forged email messages from outside the domain.) Where this may not be true is with traveling or remote users. These users may send mail from outside the domain to the STMP relay.

There are two ways to deal with this, the permissive case and the restrictive case. The permissive case is where blocking on a range of IP addresses or IP networks might make sense. For example, if the domain has an exclusive or preferred ISP and does not use others then large ISPs such as compuserv.com, aol.com and netcom.com could be blocked. If the domain had a policy that employees would not send business email from aol.com, then any mail with a sender mail address with the local domain and a client IP address originating from an aol.com domain could be safely rejected. This would not block all spam mail with a local sender address coming from the outside, but it could block the most flagrant spam relay problems.

A more restrictive approach is to only allow specific ISP networks to send mail with a sender address with the local domain. Acme.com only uses the gadget.net ISP. If the client IP address was part of gadget.net's IP network then the message would be accepted. All other local sender addresses with outside IP address would be rejected. This is the approach taken in this paper.

Restricting Senders To Specific Recipient Addresses

A final issue in dealing with spam is spammers sending to internal aliases and mailing lists. A related activity is mail bombing where a user is sent a large number of messages slowing down the SMTP relay and filling up the spool directory on the mail host. In both of these cases the solution is to use both the sender and the recipient address in deciding to accept or reject the message. If the rulesets for restricting relaying is general enough so that you can test the relationship between the sender and the recipient addresses, then these same rulesets can be used with the protected addresses as special cases of relay addresses.

It is simple to reject spam to internal aliases and mailing lists by defining a list of internal addresses which are strictly internal. If mail for one of these addresses comes from outside the domain, it should be rejected.

There are several ways to reject mail-bomb mail:

- Treat the mail-bomber as a spammer and reject all mail from that sender.
- Define a class of protected users and reject mail from selected senders to those protected users. This is the approach that Eric Allman has taken.
- Selectively match specific recipients with specific senders. This is the approach I take in this paper.

Implementation Philosophy:

The initial set of check_* rulesets I wrote were based on the simple examples given by Eric Allman. As I explained these rulesets, and their uses in class, student's questions motivated me to add additional tests.

The first improvement made was to expand the use of databases in testing the address. By using format conventions used in other sendmail databases, such as denoting user addresses with an @ sign, and using domain names and dotted IP addresses in the key, a single database can be used for a wide variety of tests. This simplifies administration.

Other techniques were borrowed from other parts of the sendmail.cf file:

- Using a lookup focus for database queries.
- Marking address to separate or identify addresses.
- Recursively calling a ruleset to test all the domains in a hostname.

Integration with the rest of the sendmail.cf file was a consideration with a special focus on filtering on a relay. This includes testing for the local domain name(s) as well as other domains the relay may handle, such as masquerade domains and virtual domains.

The last design goal was to implement these rulesets as M4 templates so that they could integrate with the M4 sendmail.cf file generation philosophy. These rulesets are implemented as a series of FEATURES for simple inclusion. User configurable parameters such as database type and location are set with define('confM4_MACRO', 'value') statements. Also specific rules are included based on whether or not a specific sendmail FEATURE or M4 macro has been set.

Databases Used By check_* Rulesets

Two databases are used by the check_* rulesets presented in this paper:

- check_mail Sender addresses to be rejected
- check_fwd Sender $| recipient pairs to be accepted or rejected

Both of these databases are simple key:value pairs stored in a UNIX ndbm or Berkeley db hashed table database.

Format Of The check_mail Database

The check_mail database allows the key to be rejected to be a user address, host or domain name, or IP network or address. The value returned as the text of the message in the SMTP error message.

The format of the check_mail database is:

- The lookup key (the address)
- The value to return

Lookup Key

The key can be one of the following:

- A specific user address like `user@host.dom`. Only this address is rejected, all other addresses from host.dom are allowed. A specific user address is any key that has an @ sign in it.
- A host or a domain name like `host01.spam.dom` or `spam.dom`. All addresses with this host or domain name following the @ sign are rejected. The mail is also rejected if this is in the MAIL From: address, or it if is in the hostname of the connecting SMTP client.
- A name like `.domain_name.x`. All host-names that end with this domain name are rejected. This rejects all mail below a domain, but not the domain itself.
- An IP network number, either one, two or three octets followed by trailing zeros, like `123.0.0.0`, or `123.123.0.0`, or `123.123.123.0`. All SMTP clients whose IP address starts with these IP network numbers will be rejected. Note that there is no check for correct class of the network entry so an entry 192.0.0.0 would reject all class C networks that start with 192.
- A specific IP network address like `123.123.123.123`. The specific SMTP client whose IP address is 123.123.123.123.

Value Returned

The value returned can be:

- The single word OK which will accept the address and stop the ruleset.
- The single word REJECT which will return a generic SMTP error message.
- A specific message for this address which will be returned as the text of the SMTP error message (This allows you to tailor your insults to specific spam sites.)

Database Example

Table 1 shows a database example.

Updating The Database

The default name of the database is: /etc/check_mail. The source file for the database can have this name since the makemap or makedbm will append the .db or .dir and .pag extensions on to this file name. The database needs to be rebuilt each time a change is made to the sourcefile.

Use the makemap command to rebuild a hashed db database. The makemap command is a standalone command to rebuild databases provided as source code in the sendmail 8 release:

```
# makemap hash /etc/check_mail \
    < /etc/check_mail
```

To rebuild a hashed ndbm database use:

```
# makemap dbm /etc/check_mail \
    < /etc/check_mail
```

or

```
# makedbm -l /etc/check_mail \
    /etc/check_mail
```

Using the -l flag on makedbm converts all of the keys to lower case.

Format Of The check_fwd Database

The *check_fwd* database is used to test if the sender/recipient pair should be allowed or rejected for SMTP relaying.

The database key is a sender and a recipient address separated by the special token '$|.' If the sender and the recipient match, then the address can be accepted or rejected. The sender and recipient address can be any of the formats allowed in the check_mail database except an IP address. The sender and recipient address can also be the special key *ALL* which will match all addresses with the other half of the key.

Like the check_mail database, the value returned can either be an error message to be returned or the special key REJECT which will return a generic error message. It can be the special key OK. If the key

Key	Function
user@host.dom	Access denied for user@host.dom
host.spam.dom	Access denied for host.spam.dom
spam.dom	Access denied for dom spam.dom
123.0.0.0	Access denied for 123.0.0.0
123.123.0.0	Access denied for 123.123.0.0
123.123.123.0	Access denied for 123.123.123.0
123.123.123.123	Access denied for 123.123.123.123

Table 1: Database Example.

returned is OK, then the sender and recipient are accepted without applying additional tests.

Database Examples:

The lookup key:

```
user1@host.dom $| user2@x.abc.com
```

would block user1@host.dom from sending to the specific user2@x.abc.com The lookup key:

```
user@host.dom $| x.abc.com
```

would block user@host.dom from sending to any user at x.abc.com The lookup keys:

```
user@host.dom $| .abc.com
user@host.dom $| abc.com
```

would block user@host.dom from sending to any user in the abc.com domain. The first key blocks mail to all hosts in the domain; the second blocks mail to the domain itself. The lookup key:

```
host.dom $| user@abc.com
```

would block any user at host.dom from sending to user@abc.com. The lookup keys:

```
.abc.com $| alias@abc.com   OK
abc.com $| alias@abc.com    OK
*ALL* $| alias@abc.com      REJECT
```

would limit senders who can send to an internal alias to users in the abc.com domain.

The first two keys allow mail from any hosts in the abc.com domain or the domain itself to the alias. The third key blocks all mail from all other domains to the alias with the generic SMTP error message "Access denied."

Benefits Of Using Databases

The benefit of the lookup keys flexibility is that a single database can be used in several places. If the key is a user address, it can be applied to a sender or recipient. If the key is a host or domain, it can be applied to the sender or recipient, but it can also be applied to the client hostname or SMTP HELO or EHLO name. If the key is an IP address or network, then it is applied to the client's IP address.

Another benefit of using a database to look up the addresses is that, as changes are made to the database, the updates are used immediately by the sendmail daemon.

Overview of check_* Rulesets

The new set of four check_* rulesets[1] allow sendmail to reject mail messages before they are delivered. The check_relay, check_mail, and check_rcpt rulesets are applied by the SMTP server during the SMTP protocol exchange with the SMTP client. The check_compat ruleset is applied after sendmail has accepted the message, but before the message is passed to a delivery agent. The results of these rulesets are used for an accept/reject decision. They are not used to rewrite the address by sendmail.

These rulesets can do anything they want. If they return a call to the $#error mailer, the message or SMTP command is rejected with the message that follows the $: part of the result, otherwise their result is ignored.

The rulesets to reject mail during the SMTP reception of a message are an important new feature because they allow mail to be rejected before it is transferred via SMTP to the destination or a relay. This off-loads the processing of the message and the generation of a bounce message by the mail host or SMTP relay.

The check_mail ruleset can be used to reject or accept the SMTP MAIL command based on the addresses given in the SMTP command as well as information about the SMTP client.

The check_rcpt ruleset is used to reject or accept a SMTP RCPT command based on the addresses given in the SMTP command. It can be used to reject mail for a specific recipient, host or domain. Since the

[1]For a short introduction on sendmail and the sendmail.cf configuration file, and rules and rulesets visit: http://www.harker.com/sendmail/overview.sendmail.html . For a short introduction on using M4 with sendmail to generate the config file visit: http://www.harker.com/sendmail/overview.M4.html .

```
LOCAL_RULESETS

# Start the named ruleset check_mail
Scheck_mail

# Put everything through ruleset S3 to focus
#   and canonicalize addresses
R$*                     $: $>3 $1

# Strip source route and pass back to ruleset
S32
R<@$+>:$+       $: $>3 $2
```

Listing 1: Defining a check_mail ruleset.

sender's address, as well as information about the SMTP client has already been determined, it can also be used to reject spam relay mail.

The check_relay ruleset is used to reject or accept a SMTP session. It is applied before sendmail accepts the TCP/IP SMTP connection. It is passed the DNS hostname for the client's IP address and the client's IP address itself separated by a '$|' (shown here folded for display purposes):

```
client.DNS.host.name $|
        client.host.address
```

If the $#error mailer is called, sendmail returns a '550 Access denied error' message for all subsequent SMTP commands.

The check_compat ruleset is different that the previous three rulesets because it is applied after the message has been accepted by sendmail, but before sendmail tries to deliver the message by calling the delivery agent. Thus, it can be applied to mail received from all sources, not just SMTP. This includes UUCP, user agents, and SMTP front-ends such as smap and smtpd. It is passed the processed sender address and the processed recipient address separated by a '$|' (shown here folded for display purposes):

```
sender@sendhost.dom $|
          recipient@rcpthost.dom
```

If the $#error mailer is called, sendmail will bounce the message as undeliverable. The drawback of this ruleset is that the bounce message is generated by the local sendmail process, not on the remote client trying to submit a message to this host.

Useful Macros in the check_* Rulesets

Sendmail 8.8 defines several macros which are useful for rejecting SMTP mail:[2]

- $&{client_name} Name of the SMTP client
- $&{client_addr} IP address of the SMTP client
- $&{client_port} Port number of the SMTP client

In these macros the SMTP client is the host trying to forward a message by connecting to this SMTP server.

Two additional existing macros are useful in the check_* rulesets:

- $&s: The hostname passed in the HELO or EHLO command.
- $&f: The resolved sender envelope address passed in the MAIL command.

[2]Use the deferred evaluation form, $&m or $&{macro_name}, to avoid having these expanded in the rule when sendmail reads the configuration file.

Implementation of check_mail Ruleset to Reject Incoming Spam

Here is the structure of check_mail ruleset:

- Focus addresses by passing to S3.
- Check non-local local with Strip_Local ruleset.
- Check local host and domain information.
- Check for valid DNS information.
- Check sender address for spam domain.
- Check macros for spam domain.

Focus Addresses by Passing To S3

The raw address is passed to the check rulesets; the addresses are unformatted and have no focus. Passing the address through ruleset S3 first put the address into the sendmail standard format:

```
user <@domain>
```

or for RFC 822 source route address:

```
<@domain> : balance of source route
```

This is called the 'focused address' with the next host in line for delivery inside the focus angle brackets, <>. In this configuration it is assumed we are not interested in the source route address, only in the final destination, so RFC 822 source route addresses are stripped.

Listing 1 shows how to define the check_mail ruleset with these rules in a sendmail.cf file generated with M4.

The M4 macro LOCAL_RULESETS includes the text that follows it in the sendmail.cf file before mailer definitions and is used for all ruleset additions in this paper.

Testing for local host or domain information:

A simple performance benefit is to accept addresses with local host or domain information. This avoids any additional processing of these addresses and eliminates their database lookups.

This local information is stored in the sendmail macros and classes:

- $w: My hostname
- $=w: Other host I accept as local
- $m: My domain name
- $=M: Domains I masquerade

The matching of these macros and classes is done with a repeated series of Domain Match Blocks (my terminology)

Domain Match Block

Listing 2 shows the rules that perform domain matching. On the Left Hand Side (LHS) of the rule, the match.pattern inside the focus of the address can

```
# Does it end with our domain, if so OK
R$+<@ match.pattern . >        $@ OK
R$+<@ $+ . match.pattern . >   $@ OK
```

Listing 2: Domain matching rules.

be a macro, $m, class, $=w, or text, abc.com. The first rule tests for the match.pattern itself. The second rule tests for all hosts and sub-domains below the match.pattern.

If the address matches the LHS pattern, then the RHS exits the ruleset and returns the symbolic name OK. The OK returned by these rules are strictly for human consumption. A ruleset returning OK shows that the address was explicitly accepted by a rule. If the address itself is returned by the ruleset, then the address fell out the bottom of the ruleset. In both cases since the $#error mailer is not returned by the ruleset, the result is discarded by sendmail and the address is accepted.

Rejecting Addresses Hidden Behind Our Domain

We want to avoid addresses hidden behind our hostname or domain name:

```
spammer%spam.dom@my.dom
```

Since this check is something we will want to do in several of the check_* rulesets, we create a custom StripLocal ruleset to do it. After calling ruleset S3, we pass all address to the StripLocal ruleset:[3]

```
# Check that the address is
# local, not user%host@my.dom
R$*<@$+> $:$>StripLocal $1<@$2>
```

The StripLocal ruleset tests for domains I do masquerading for, $=M, as well as domains looked up in the genericfrom database, $=G, and other hostnames I will treat as local, $=w.

Standard Block

Listing 3 shows the standard block of rules. The first rule tests if a single token is followed by the domain class:

```
< @ $=M . >
```

then the token passed to the dequote database, in case the sender passed a quoted string. The dequote database is a pseudo database built into sendmail which removes quotes from a string and parses the results into tokens.

The second rule tests if the focus is preceded by a non-local separator, $=O[4] and if the address ends with the domain class. If this is true, the address is not truly local so the domain is stripped and the user portion of the address is passed back to the $>StripLocal ruleset.

The third and fourth rules do the same thing except that the domain class is tested with preceding host or sub-domain names:

[3]About $>StripLocal $>3 RHS syntax: When two rulesets are specified on the RHS, the first is called, and then the second is immediately called before executing the first rule of the first ruleset. This means that the second ruleset is processed first and the output of the second ruleset is used as the input to the first ruleset.

[4]The operators '!', '%', and '@' are the token separators contained in class O.

```
# Does it end with another domain we are known as?
#   make sure it is local
R$- < @ $=M . >              $: $(dequote $1 $) < @ $2 . >
R$* $=O $* < @ $=M . >       $@ $>StripLocal $>3 $1 $2 $3
R$- < @ $+. $=M . >          $: $(dequote $1 $) < @ $2 $3. >
R$* $=O $* < @ $+. $=M . >   $@ $>StripLocal $>3 $1 $2 $3
```

Listing 3: Standard block of rules.

```
LOCAL_CONFIG
Kcheck_mail type -o -a.REJECT /etc/check_mail
```

Listing 4: Defining check_mail in sendmail.cf

```
# Lookup user@host.dom
R $+<@$+.>  $: $(check_mail $1@$2 $: $1<@$2.>$)
```

Listing 5: Looking up entire address in check_mail database.

```
$:              Apply the rule once, do not recurse
$(check_mail    Start of the lookup in the check_mail database
$1@$2           Key used in the lookup
$: $1<@$2.>     Default rewrite if lookup fails
$)              End the lookup
```

Listing 6: The RHS syntax.

```
< @ $+. $=m . >
```

The StripLocal ruleset does not reject mail, it only removes un-needed local information. The StripLocal ruleset is applied before the preceding local checks.

Rejecting Addressees in the check_mail Database

With the address in a stripped focused format, it can be checked against the check_mail database in order to test if it should be rejected.

Before the database can be used, it must be defined in the sendmail.cf file: Listing 4 would define the check_mail database in the beginning of a sendmail.cf file generated with M4. The K line defines the database check_mail to be a database type (hash or dbm) which is located in /etc/check_mail. The -a.REJECT flag causes sendmail to append the pseudo domain .REJECT to the value returned if the look-up is successful. If the lookup fails, the look-up key is returned unmodified unless a default rewrite is specified in the RHS of the look-up rule.

With the database defined, the rule in Listing 5 looks up the entire address in the check_mail database. Listing 6 shows the RHS syntax.

Since the keys stored in the database do not have focus, the lookup key used is the user@host.domain address without focus or trailing dots. If the database lookup is successful, then the associated value is returned with the pseudo domain .REJECT appended, which is used to reject the address. If the lookup fails, a default rewrite is used to put the focus and trailing dot back into the address.

Once the address has been looked up, the next step is to test if the address should be rejected by testing if the address now ends with .REJECT If it does, use the string that precedes the .REJECT as the error message returned by the error mailer:

```
# if address ends with .REJECT
# use value returned as the
# error message
R$+.REJECT          $#error $:$1
```

Two additional tests that are made are for the fixed values OK.REJECT and reject.REJECT.[5] A return value in the database of reject.REJECT will reject the address with a default error message. A return value of OK.REJECT will accept the address without any additional processing. The ruleset will exit with the symbolic address OK.

These three rules create a block of rules which is referred to as a Reject Block of Rules (my terminology). The Reject Block is used after every database lookup in the check_* rulesets:

```
# if name is OK.REJECT,
# exit ruleset with OK
ROK.REJECT                  $@ OK

# if name is reject.REJECT
# use default error message
# (shown folded)
Rreject.REJECT
        $#error $:Access denied

# otherwise use value returned
# as the error message
R$+.REJECT          $#error $:$1
```

Checking Macros Against The check_mail Database

In addition to the address itself, several macros are available for checking against the check_mail database. In particular:

$&{client_name}	The client hostname from the socket
$&s	The hostname from the HELO or EHLO Command

The separate look-up focus technique introduced in sendmail 8 is used to test these macros.

sendmail 8 Double Focus

sendmail 8 uses a double focus technique which allows a portion of an address to be checked against a database without modifying the original address. This is used in the mailertable, genericstable, and virtusertable FEATUREs. The double focus is generated by replicating the host or domain name in the focus in an initial lookup focus

```
R$* <@$+> $*    <$2> $1 <@$2> $3
```

The first focus is a lookup focus, with just the host or domain name. The second focus is the original domain focus with an @ sign followed by the host or domain name (shown here folded):

```
<lookup focus>
     stuff <domain focus> stuff
```

The lookup focus is used in a database lookup (also folded):

```
R < $+ > $* <@$+> $*
  <$(dbname $1 $)> $2 <@$3> $4
```

The result in the lookup focus can be checked for a specific pattern. If it does not match, the host or domain name in the lookup focus can be rewritten and looked up in the database again.

If it still does not match the lookup focus can be stripped to leave the original address

```
R<$+> $* <@$+>$*      $2 <@$3> $4
```

Checking A Macro with a Lookup Focus

The current macro value is prepended in a look-up focus in the address.[6] As the macro is put into the

[5]Remember, sendmail is case-insensitive in the rulesets.

[6]Remember, the original address will not start with a focus since all source route addresses have been previously stripped.

look-up focus, it is passed to the dequote database to break the macro into separate tokens for testing. This value is then looked up in the database and the result checked with a Reject Block of Rules. Finally, if the pseudo domain .REJECT is not returned, the look-up focus is stripped leaving the original address [FN.X]; see Listing 7.

Walking a Fully Qualified Domain Name

The tests so far have tested the entire lookup key. This will reject specific hosts or a specific domain, but it will not reject all of the hosts from a rejected domain. In order to do this the technique of walking the domain name is borrowed from the mailertable FEATURE of sendmail 8. A Fully Qualified Domain Name (FQDN) is passed in a lookup focus to a separate ruleset that recursively calls itself stripping the first token of the FQDN until a match is found or until only the final token remains of the FQDN.

The CheckMailDB ruleset is used to check the host part of the address. Both a specific hostname and hosts, and sub-domains below a domain can be rejected by walking the hostname. This ruleset checks the host/domain name in the lookup focus. A Reject Block of Rules is used to check the result. If a match is not found, then the first token is stripped and the domain preceded by a dot is looked up. Again, a Reject Block of Rules is used to check the result, and if a match is still not found, the remaining domain name without the preceding dot is passed back to the top of the same CheckMailDB ruleset. This cycle of stripping the first label of a domain name and calling the same ruleset again continues until only a single token remains. If the hostname makes it to the end of this process, then neither the hostname or any of its parent's domains have been blocked, and the recursive call to the ruleset unwinds as each ruleset call returns to the previous ruleset call; see Listing 8.

The flow of the successive calls to the CheckMailDB ruleset is as follows:

```
1st call CheckMailDB input:
        <host.sub.dom> org_addr
  Look-up host.sub.dom
  Lookup .sub.dom
  2nd call CheckMailDB input:
             <sub.dom> org_addr
    Look-up sub.dom
    Lookup .dom
    3rd call CheckMailDB input:
                <dom> org_addr
      Look-up dom
      Only one token
    3rd call CheckMailDB returns <dom>
  2nd call CheckMailDB returns <dom>
1st call CheckMailDB returns <dom>
```

The CheckMailDB ruleset is invoked in the check_mail ruleset by putting the hostname in a lookup focus and calling the ruleset. If the address

```
# Put the macro in a look-up focus
R$*             $: <$(dequote "" $&{macro.name} $)> $1

# look-up the macro in the check_mail database
R <$+> $+       $: $(check_mail $1 $: <$1> $2 $)

{A Reject Block of Rules}

# strip the lookup focus if .REJECT was not found
R< $* > $*      $: $2
```

Listing 7: Look-up focus followed by revert to original address.

```
SCheckMailDB
# first lookup the name as a hostname
R <$+> $+               $: $(check_mail $1 $: <$1> $2 $)

{A Reject Block of Rules}

# does the lookup name still have two or more tokens
#   separated by a dot?
# Strip the first token and lookup .domain
R< $- . $+ > $+         $: $(check_mail .$2 $: <.$2> $3 $)

{A Reject Block of Rules}

# If name start with a dot, strip the first dot and pass
#   remainder of the domain back to CheckMailDB ruleset
R< . $+ > $+            $@ $>CheckMailDB <$1> $2
```

Listing 8: Unwind domain name looking for rejects.

still has a lookup focus when it is returned by the CheckMailDB ruleset then it did not find a match in the database and the lookup focus can be stripped; see Listing 9.

In addition to walking the hostname in the address, the SMTP client hostname, $&{client_name}, and HELO or EHLO name, $&s, can also be walked by passing them to the CheckMailDB ruleset.

Rejecting Bad DNS Names

A lot of spammers use invalid sender addresses to avoid error messages and flames being sent back to them. A simple test to reject this type of mail is to check that the hostname or domain in the address is valid. This can be done by passing the address through ruleset S3 and then checking for a trailing dot. Ruleset S3 adds a trailing dot to the end of all hostnames that are found in DNS. The test can be made with the exclusive class match against class dot, $~., which contains a dot by itself; see Listing 10.

This simple test may be too heavy-handed; not all hostnames are registered in DNS. Many sites limit what internal hostnames are listed in their DNS databases or have poor administration of their DNS databases. A better test would be to walk the domain looking to see if the hostname, its sub-domain, or parent domain are valid hostnames. Since the lookup is

```
# Check the hostname and domain name against the
#   check_mail database putting the hostname
#   without the trailing dot in a lookup focus
R$* <@ $+. >            $: $>CheckMailDB <$2> $1 <@$2.>

# lookup focus no longer needed, strip it off
R< $* > $*              $: $2
```

Listing 9: Checking host and domain names against database.

```
# if the hostname in the focus does not end with a
#   dot, reject it.
R$* <@ $+ $~. > $#error $:$2.$3 unknown to DNS
```

Listing 10: Checking against the 'dot' class.

```
SWalkDNS
# Query DNS for domain in the lookup focus
R< $- . $+ > $*         $: < $[ $1 . $2 $] > $3

# Was the domain in the lookup found in DNS and
#   now ends with a dot?  If so, exit the ruleset
R< $+ . > $*            $@ < $1 . > $2

# Otherwise strip 1st token, and pass back to WalkDNS
R< $- . $+ > $*         $@ $>WalkDNS <$2> $3
```

Figure 11: Query DNS for valid hostnames.

```
# Pass non-canonicalized addresses to ruleset WalkDNS
# We strip the first token since S3 ( S96 really)
#   has already looked up the name.
R$* <@ $+ . $~. >       $: $>WalkDNS <$3> $1 <@$2 . $3>

# Is the hostname not part of a valid domain name
# i.e. does the name in the lookup focus still not
#   end with a trailing dot
R<$* $~.> $+ <@$+>      $#error $: Access denied, $4 unknown to DNS

# The hostname was part of a valid domain name
# Strip off the Qualified Domain Name in lookup focus
# adding trailing dot to hostname so all hostnames end
#   with a dot
R< $* > $+ <@ $+>       $: $2 <@$3.>
```

Figure 12: Calling WalkDNS and checking result.

in the DNS database, not the check_mail database, a separate WalkDNS ruleset is defined; see Listing 11.

The rules that call the WalkDNS ruleset and check the result are shown in Listing 12.

Rejecting Mail from Specific IP Addresses

IP addresses are used as keys in the check_mail database. These keys can be an IP network number:

```
123.0.0.0
123.123.0.0
123.123.123.0
```

or a specific IP network address:

```
123.123.123.123
```

The client's IP Address is taken from the socket and is stored in ${client_addr}; see Listing 13.

Implementation Of check_rcpt Ruleset To Reject Relay Mail

The structure of the check_rcpt ruleset follows the structure of the check_mail ruleset. Much of the checking for local user and host information is the same, with the exception that the recipient address passed in the SMTP RCPT command is being tested, rather than the sender address. If the recipient ends with one of the host or domains for which we are a relay then the address is accepted. The check_rcpt ruleset differs from the check_mail ruleset when the address is not local. It accepts the address if the SMTP client's IP address is within the local domain. The other main difference is that the sender address from the SMTP mail command is prepended to the address, separated by a $|, and the sender $| recipient pair is checked against the check_fwd database for accepting/rejecting. Structure of *check_rcpt* ruleset:

- Focus addresses by passing to S3
- Check non-local local with Strip_Local ruleset
- Check *sender* **$|** *recipient* pair for accepting/rejecting
- Check local host and domain information
- Check for valid DNS information
- Check for local client IP addresses

Checking For Local Client IP Addresses

To check if the SMTP client is a host with a local IP address, the client's IP address is put into a lookup focus and compared to local IP network numbers and addresses. Since the test is for accepting a non-local recipient from a client and most of the matching will be for entire IP network numbers (which is typically somewhat limited) a class match can be used. The class match does not need a separate file if the domain only has a few class B or C networks. If the domain needs to accept mail for lots of IP addresses or network numbers a file class definition can be used.

The only drawback of using a class is that the sendmail daemon needs to be killed and restarted each time a change is made. The IP address in the class definition can either be a dotted quad, 123.123.123.123, in which case it matches the exact IP address, or it can be one, two, or three octets with out the trailing zeros, 123, 123.123, or 123.123.123, in which case it matches all IP addresses that start with the network number.

```
# Tack on client IP address in a lookup focus
# Note: no addresses should start with focus since all
#   src routes have been stripped
R$*             $: <$(dequote "" $&{client_addr} $)> $1

# Reject specific IP addresses (all four octets)
R< $-.$-.$-.$- > $*             $: < $(check_mail $1.$2.$3.$4 $) > $5

{A Reject Block of Rules}

# Reject Class C networks (first three octets)
R< $-.$-.$-.$- > $*     $: < $(check_mail $1.$2.$3.0 $: $1.$2.$3.$4 $) > $5

{A Reject Block of Rules}

# Reject Class B networks (first two octets)
R< $-.$-.$-.$- > $*     $: < $(check_mail $1.$2.0.0 $: $1.$2.$3.$4 $) > $5

{A Reject Block of Rules}

# Reject Class A networks (first octet)
R< $-.$-.$-.$- > $*     $: < $(check_mail $1.0.0.0 $: $1.$2.$3.$4 $) > $5

{A Reject Block of Rules}

# Strip off IP address in lookup focus
R< $* > $*              $: $2
```

Listing 13: IP address manipulation.

The class can be defined with:

```
# Define a class of local IP
# addresses and networks in the
# sendmail.cf file itself
C{LocalIP}192.102.231
```

or

```
# Define a class of local IP
# addresses and networks from
# a file
F{LocalIP}/etc/LocalIP
```

Once the class is defined it can be tested with:

```
# Tack on client IP address in
# a lookup focus. No addresses
# should start with focus since
# all src routes have been
# stripped
R$*     $: <$(dequote ""
           $&{client_addr} $)> $1
# (above line folded)

# Accept our IP networks
R< $={LocalIP} .$+ > $*     $@ OK
# Accept specific IP addresses
R< $={LocalIP} > $*         $@ OK

# Strip off IP address in lookup
# focus
R< $* > $*                  $: $2
```

The rule

```
R< $={LocalIP} .$+ > $*     $@ OK
```

will match either the first, first.second, or first.second.third octets of the IP address using *sendmail*'s rule matching feature of starting with one token and extending the number of tokens compared until a match is found.

Checking *sender* **$|** *recipient* Pair for Accepting/Rejecting

The main part of the *check_relay* ruleset is checking the *sender* **$|** *recipient* pair to see if they should be accepted or rejected. This follows checking the check_mail database in the check_mail ruleset, but it has an added twist in that there are two addresses, a sender and a recipient, and both need to have their domains walked. For each sender address, hostname, or domain name, a special key, sender_addr $| *sender*, is used to avoid excessive database queries. This special key is checked to see if the sender has any blocking. If the sender does not have this key, then no further checking of the sender $| recipient pair is done. If the sender does have the key, then the sender and recipient addresses are checked.

The walking of both the sender and the recipient address is done in two steps. The sender's address is walked and if the complete address, or the host or domain portion of the address has blocking, then the recipient address is then walked with the blocked sender portion. The advantage of this approach is that if a sender has no blocking, then the recipient portion of the address is not checked against the database.

Walking the Sender Address

In order to walk the sender address, the macro S&f is inserted before the recipient address in a lookup focus.

```
# prepend the sender's name in
# a lookup focus
R$*                 $: <$&f> $1
```

With the sender in a lookup focus, the sender and the special key ***sender*** are checked in check_fwd db to see if sender has blocking:

```
$(check_fwd $&f : *sender* ...
```

If this pair matches the value with .REJECT appended is returned. Otherwise, the sender is kept in the lookup focus so we can continue testing the sender in the lookup focus using constructs like '`$: $1 $)`' in contexts as shown in Listing 14. If the lookup was successful, the returned address is:

```
<string.REJECT> user <@rcpt.dom>
```

If the returned value is OK.REJECT, then sender explicitly is accepted

```
R<OK.REJECT> $+          $@ OK
```

If the lookup focus ends with .REJECT, the sender has blocking and we look up the sender and the recipient *check_fwd* db like this:

```
$(check_fwd $&f $| $2 @ $3 ...
```

If this pair matches, the value is returned with .REJECT appended. Otherwise, we rewrite as

```
<sender $| rcpt hostname> rcpt addr
```

using a construct somewhat like this:

```
... $: < $&f $| $3 > $2 <@ $3 > $)
```

in a specific context as in Listing 15.

We do not reject the address immediately. Instead we check if it is still as *sender* **$|** *recipient* pair. If it is, the sender has blocking, but it is not for the specific user@rcpthost.dom address. We walk the

```
# Reject from a specific user
# We lookup the sender and the special key *sender* in
#   check_fwd db to see if sender has blocking
R<$+> $+        $: < $(check_fwd $1$|*sender* $: $1 $) > $2
```

Listing 14: Rejecting mail from a specific user.

recipient hostname to see if it is the specific user@senderhost.dom address that has blocking; see Listing 16. When the Walk_Rcpt ruleset returns if a match was found, the value in the lookup focus will end with .REJECT, otherwise it will be the sender's address by itself. We check it for a match with a reject block of rules.

What we have done at this point is checked the complete sender address against the recipient address, hostname, and domain name. If a match is not found, the lookup focus contains the full sender addresses. The next step is to walk the sender's hostname to see if the sender's host or domain has blocking.

```
# Walk the sender's hostname
# looking for an accept/reject
# Put the recipient addr back
# into lookup focus
R<$+@ $+ > $+
        $: $>Walk_Send < $2 > $3
# (folded here)
```

Like the Walk_Rcpt ruleset, when the Walk_Send ruleset returns, if a match was found, the value in the lookup focus will end with .REJECT. We check it for a match with a reject block of rules.

If a match is not found, then the sender did not have blocking for this recipient and the lookup focus is stripped.

The Walk_Rcpt Ruleset

The Walk_Rcpt ruleset is similar to the Check-MailDB ruleset in that it recursively calls itself each time stripping the first token and dot from the recipient hostname which follows the **$|**. in the lookup focus. Listing 17 shows an example.

The Walk_Send Ruleset

The Walk_Send ruleset follows the same steps of checking the full sender address except that the address in the lookup focus is the sender's host or domain name, not a user@host.dom address.

Like checking the full sender, the first step is to check if the sender's host or domain name has blocking; see Listing 18.

If the lookup was successful, the returned address is:

```
string.REJECT <send.host.dom>
                    user<@rcpt.dom>
```

```
# If this pair matches, reject
# Otherwise, put the sender and recipient
#   in the lookup focus
R<$+.REJECT> $+<@$+>     $: $(check_fwd $&f$|$2@$3 $: <$&f$|$3> $2<@$3> $)
```

Listing 15: Checking senders and recipients after 'REJECT'.

```
# If lookup focus has a $| then sender still has blocking, walk the hostname
R<$+$|$+> $+             $: $>Walk_Rcpt <$1$|$2> $3
```

Listing 16: Walk recipient hostname to checking blocking.

```
SWalk_Rcpt

# first lookup the name as a hostname
R <$+$|$+> $+            $: $(check_fwd $1$|$2 $: <$1$|$2>$3 $)
# If result ends with .REJECT, we found a match,
#   now exit Walk_Rcpt
R $+.REJECT              $@ $1.REJECT

# If lookup focus has a $| then sender still has blocking,
#   walk the hostname
R< $+ $| $-.$+ > $+      $@ $>Walk_Rcpt <$1 $| $3> $4

# If we have do not have two or more tokens, strip recipient from LUF
R< $+ $| $+ > $+         <$1> $3
```

Listing 17: Recursively strip tokens for checking.

```
SWalk_Send

# first lookup the sender name as a hostname with *sender* special key
R<$+> $+         $: $(check_fwd $1:*sender* $:  $) <$1>$2
```

Listing 18: Checking for blocking.

Notice that the reject message is before the lookup focus. This is to keep the sender's host or domain name intact so it can be passed to Walk_Rcpt.

If the lookup fails, the sender host or domain name did not have blocking and the returned address is:

```
<send.dom> user<@rcpt.dom>
```

We strip the first token in the lookup focus and check the *.domain* name against the *check_fwd* database; see Listing 19. If this lookup fails, we strip the first dot and call the *Walk_Send* ruleset again

```
# If address starts with a
# lookup focus with 3 or more
# tokens, sender host not found,
# walk the sender hostname
R<.$+> $+
       $: $>Walk_Send < $2 > $3
```

If at any point the database query is successful in the *Walk_Send* ruleset and lookup focus is preceded by a string.REJECT then the sender host or domain name had blocking and we need to check the recipient address and the recipient's host or domain name for explicit blocking. See Listing 20.

Validating the check_* Rulesets

These ruleset can be checked with address test mode, using `sendmail -bt`. Testing the sender in the check_mail ruleset and the recipient in the check_rcpt ruleset is as normal. You call the ruleset with the address:

```
check_mail user@spammer.dom
check_rcpt user@abc.com
```

To check a macro value or the sender in the *check_rcpt* ruleset, the macro must be set first with:

```
.D{client_addr}123.123.123.123
.Dfuser@spammer.dom
```

The ruleset is then run as normal.

Modified checksendmail

The *perl* script *checksendmail* is a program to automate the exhaustive testing of a list of addresses. It is a very useful debugging tool for *sendmail*. and was originally written by Gene Kim, Rob Kolstad, & Jeff Polk. I have modified it with two new flags:

- **-cm**: Check addresses against the check_mail ruleset
- **-cr** Check addresses against the check_rcpt ruleset

The address file has also been expanded to allow the inclusion of *sendmail* debugging macro definitions starting with **.D**to allow a sequence of addresses to be tested with specific macros set to specific values before the address is passed to the ruleset. This is very useful for doing regression testing against a series of addresses.

This modified version of checksendmail is available from: http://www.harker.com/sendmail/checksendmail. html .

Effectiveness Of *check_** Rulesets

The rulesets presented are currently in use at Pacific Bell and Harker Systems. They are relatively effective. On a technical basis, they are very effective blocking all the addresses, host and domain names, and IP addresses that are defined in the databases. The anti-relaying rules are very effective blocking spammers using a site as a spam mail exploder. On a practical basis, the anti-spam rules are less than effective. For spammers who send spam from a consistent host or domain, the ruleset is works well. Unfortunately, much of the spam is sent from throw-away accounts at large ISPs with liberal trial period policies. By the time the site is aware of incoming spam from one of these accounts, the spammer has already moved on to a new ISP account.

If a spammer is targeting a lot of users at a site and the spam can be detected quickly, then the rulesets can be very effective in blocking subsequent spam from that user.

```
# If address starts with a lookup focus with 3
#   or more tokens, sender host not found,
#   walk the sender hostname
R<$-.$+> $+          $: $(check_fwd .$2:*sender* $:  $) <.$2>$3
```

Listing 19: Strip the first token in the focus and try again.

```
# Lookup sender and full recipient
R$+.REJECT <$+> $+<@$+>    $: $(check_fwd $2:$3@$4 $: <$2:$4> $3<@$4> $)
sp 0.5
# does the lookup name still have sender and recipient?
# If so strip the first token and dot and pass remainder
#   of address back to Walk_Rcpt ruleset
R< $+ : $- . $+ > $+       $@ $>Walk_Rcpt <$1:$2.$3> $4
```

Listing 20: Sender has blocking; check recipient.

Conclusion

The rulesets presented here are designed to filter out unwanted email. Unfortunately, these techniques have limited success because it is a reactive battle and the spammers can keep one step ahead in the game. Choosing what sites to block requires careful consideration, so that valid email from a problem domain is not accidently rejected. The selected blocked sites need to be documented and agreed to by management. These tools are useful, and as more sites implement anti-spam and anti-spam relay rulesets, there will be fewer sites that the spammers can abuse.

Availability

The most current version of this discussion and rulesets will be found at: http://www.harker.com/sendmail/anti-spam .

Acknowledgments:

Thanks to Eric Allman, one of the great unsung heroes of the Internet. Email is THE "Killer Application" on the Internet. Sendmail makes Email on the Internet work, and Eric Allman wrote *sendmail* and has been maintaining it all these years.

Author Information

Robert Harker is a Senior Consultant with Harker Systems and responsible for teaching "Managing Internet Mail" (formerly "Sendmail Made Simple," and "Advanced Sendmail and Electronic Mail Domains") and consulting for a variety of clients. Specializing in Internet electronic mail, he has over 14 years of UNIX and networking experience, and has built and managed networks for technology and transportation companies such as DHL, National Semiconductor, and Motorola. Mr. Harker has over 7 years of consulting and teaching experience, including a comprehensive series of TCP/IP networking classes given through the University of California Extension Program. His electronic mail address is harker@harker.com His web page is www.harker.com Reach him via U. S. Mail at:

Robert Harker
Harker Systems
1180 Hester Ave.
San Jose, CA 95126
harker@harker.com
408-295-6239

References

Eric Allman, "Installation and Operation Guide For Sendmail Version 8.8," Sendmail.ORG, 1997.

Eric Allman, "Anti-Spam Provisions in Sendmail 8.8," Sendmail.ORG, 1997.

Claus Assmann, "Using a database in the *check_** rulesets," Christian-Albrechts-University of Kiel, 1997.

Robert Harker, "Managing Internet Mail: Setting Up and Trouble-shooting sendmail and DNS," Harker Systems, 1997.

A Better E-Mail Bouncer

Richard J. Holland – Rockwell Collins, Inc.

ABSTRACT

This paper describes a portable electronic mail bouncer which sends detailed information back to the sender when a mail message can not be delivered to it's intended recipient. The bouncer was originally written to handle a large merger between multiple DNS domains, and is implemented entirely in Perl5 as a mail delivery agent. The bouncer operates under the concept of "least privilege" so it's safe to run directly from mail transport agents such as sendmail. The bouncer is designed to make the human processes and interactions in dealing with undeliverable E-mail easier for both postmasters and end-users alike.

Introduction

Local Background

Most large networks are in a constant state of flux when it comes to account management and electronic mail routing. New users are added daily as old users are removed, often without thought of potential future consequences in an ever expanding electronic world. As more and more computer neophytes start using the Internet, handling mail delivery problems and bounces will become a much larger problem for any large site's postmaster.

Like postmasters at most sites with several thousand users, our bounced mail is run through multiple filters in an attempt to either auto-respond to problems or to sort the problems into related issues for easier handling. The rapid increase in the amount of unsolicited commercial E-mail is making this even more important. This paper examines the other side of these issues – what an end-user sees in an undeliverable message, rather than what a postmaster sees. Hopefully by improving the end-user interface, we will lower the number of undeliverable messages a postmaster must deal with directly.

This spring, Collins Avionics & Communication Division merged with it's sister company, Collins Commercial Avionics, to form Rockwell Collins, Inc. The new combined enterprise network contains over 10,000 active network users, and several thousand old accounts which no longer exist, but whose userid's can not be re-used in an effort to prevent mis-delivered electronic mail. One of the first steps in merging networks of this size are to correct any namespace collisions, both hostnames and login IDs. Forcing uniqueness of existing usernames generally isn't looked upon favorably by the person whose address is changing; they may have their old address printed on business correspondence and recorded in thousands of "From:" headers scattered across the Internet. In an enterprise network, even a collision rate of less than 5% can make a user-friendly solution a value-added task. In an effort to help our users and customers do business more efficiently, we're notifying senders when an address changes with a clearly written explanation of what happened and why. This bounced message gives the sender the recipient's new address, similar to the sendmail redirect [1] feature, but with more detailed information.

Why E-Mail is Bounced

In most cases it is possible to use alias maps on mail hubs and gateways to re-route electronic mail to the user's new address automatically, but usually the user has no way to control how long this forwarding is enabled. When it is removed, the old address suddenly generates undeliverable messages with brief errors such as those shown in Listing 1. While this tells the user why the message was returned, it doesn't explain why the username they're sending to is now unknown,

```
... while talking to mailhost.domain.com.:
>>> RCPT To:<user@mailhost.domain.com>
<<< 550 <user@mailhost.domain.com>... User unknown
550 user@mailhost.domain.com... User unknown
```

Listing 1: Undeliverable message error.

```
... while talking to mailhost.domain.com.:
>>> RCPT To:<user@mailhost.domain.com>
<<< 551 User not local; please try <user@elsewhere.com>
551 User not local; please try <user@elsewhere.com>
```

Listing 2: Forwarding address in return message.

nor does it suggest any corrective actions. While the above example makes perfect sense to any Postmaster or System Administrator, many end-users simply don't understand how to interpret many computer-generated error messages like this, and then must contact a System Administrator or attempt to contact the person they're trying to send the E-mail to via alternate means to get the new E-mail address.

Newer versions of sendmail (Berkeley v6.25 and later) can be configured to take advantage of a feature called redirect which is used to provide forwarding addresses in the returned message, rather than just saying "User unknown." A sample redirect bounce might look like Listing 2. Once again sufficient for System Administrators, this time the error message communicates the recipient's new address. However, these terse error messages often confuse end users who may wonder if this is just informational and whether or not their message was delivered to the new address provided or not.

Bouncer Design Considerations

With a user community of over 10,000 people, we wanted to develop a methodology for returning undeliverable messages to senders with concise, plain English explanations of why the message is being returned, and suggest corrective actions they should take to ensure their message is delivered successfully in the future. The bouncer needs to handle many potential reasons for a userid change such as namespace collisions, legal name changes, employees who have left the company, etc. In order to facilitate rapid communications, in addition to returning a mail message to the sender, where possible an attempt is made to deliver the message to the user's new address if permitted by local security policies.

The final implementation language was also given careful consideration. First and foremost, because of our migration schedules we needed a language which would facilitate rapid prototyping and development. Ideally, the language would also allow us to easily secure the bouncer. Perl [2] meets both of these requirements for several reasons:

- Perl scripts are usually shorter than comparable shell scripts or C programs, and are therefore easier to maintain. The entire bouncer program is under 1,000 lines of code, including comments.
- Most functions the bouncer needed to perform could be handled with native perl functions, which means we didn't need to pass user input to a shell, with potentially disastrous results. We could also take advantage of Perl's *taintperl* program, which considers any user-supplied data as "tainted" and therefore unsafe for shell interpretation.
- Perl's unmatched regular expression capabilities were the final deciding point. Since a good portion of the bouncer's task is to parse text (configuration files and E-mail messages) and perform actions based on recipient and sender information, Perl was a natural choice for a fast implementation which is easy to maintain by even junior System Administrators.

Implementation & Configuration

Filter Implementation

The initial implementations of the bouncer program ran with minimal privileges as a simple mail filter. Users who were to have their mail bounced would receive an alias similar to:

```
oldname: newname,bouncer or
oldname: bouncer
```

In this way, the recipient would receive a copy of the message if permitted by local security restrictions, and the bouncer account would also receive a copy. The bouncer account contained a simple .forward file which reset IFS for security purposes, executed the bouncer script, and if it failed, returned an exit status of 75 so that sendmail would bounce the message as it normally would have. The .forward file read:

```
|IFS=' ' & exec /path/bouncer.pl\
    || exit 75 #bouncer
```

The first problem with this implementation is that mailing list distribution programs write their headers in many different ways. There doesn't seem to be any standard as to which header will contain the mailing list address, and which will contain the mailing list maintainer's address.

The other problem with the filter implementation is that if the original recipient appears only in a Bcc: header, the recipient is hidden from the filter; only the delivery agent knows who to deliver the message to. By the time the message reaches the bouncer, the Bcc: headers have been removed for privacy reasons by the mail transfer or delivery agent.

Delivery Agent Implementation

Because of these problems, the bouncer was rewritten as a mail delivery agent. This required the addition of a few lines to the mail system's sendmail.cf configuration file, and a re-write of the bouncer code for security reasons. At the same time, the code was moved to the mail server's local file systems, rather than the bouncer account's home directory (in fact, the bouncer account is no longer needed; the delivery agent can be run as any unprivileged user, such as "nobody"). Because the bouncer is running as a delivery agent, it must run under the same assumptions a SUID [3] program would run, since it's launched via sendmail with system privileges. Since the bouncer doesn't actually have to do any local mail delivery, it can relinquish these privileges immediately (running instead as "nobody"), which minimizes the risk of security issues. In addition, it carefully "untaints" all user input that is passed back to

external programs, to prevent a shell from interpreting malicious data.

Configuration of the delivery agent requires the addition of two lines to the sendmail.cf file. The first addition is a mail delivery agent command line, such as:

```
Mbouncer, P=/bin/bouncer,
    F=DFMPlms S=10, R=20,
    A=bouncer -a $f -d $u
```

This line fully defines the operation of the bouncer program and it's interaction with sendmail. The F= flag tells sendmail that the bouncer needs a Date: header (D), a From: header (F), a Message-ID: header (M), and a Return-Path: header (P). It also states that /bin/bouncer is a local delivery agent (l), that multiple recipients are permissible (m), and to strip quotation marks (s). The S= flag tells sendmail to process the sender's envelope and header addresses with ruleset 10 (after ruleset 3 and 1, but before ruleset 4). The R= flag tells sendmail to process the recipient's envelope and header addresses with ruleset 20 (after rulesets 3, 0, and 2, but before ruleset 4). The A= flag declares the command line arguments the bouncer will see in it's argv[] array. In this case, sendmail will pass the sender's envelope address after the -a option ($f macro) and will pass multiple recipients after the -d option ($u macro).

Once the delivery agent has been established, ruleset 0 must be modified in order to enable it. If addresses of the form *user@bouncer* are to be handled, the line in Listing 3 is added to ruleset 0. If addresses of the form *user@newdomain.com.bouncer* are to be handled, the line in Listing 4 is added to ruleset 0. Which of these rules are used is at the discretion of the postmaster implementing the bouncer; either or both (or other similar configurations) can be used, depending on the local site's configuration preferences. For example, if there is already a host named bouncer in the new domain, the second address specification would be preferable to avoid confusion. The bouncer program will handle both cases, as ignores everything after the username portion of the address.

Functionality & Configuration

The bouncer code has several configurable options within the code itself. These options are all defined at the beginning of the program and allow the administrator to set:

- Who receives E-mail/pages if the bouncer fails in some unforeseen way?
- Who should the bouncer deliver mail as (e.g., MAILER-DAEMON, postmaster, etc.)?
- Who should replies to bounced messages be delivered (e.g., helpdesk, postmaster, etc.)?
- Which Precedence: headers indicate a message that should not be bounced?
- If a message isn't bounced, is it delivered with an addendum to the original recipient?
- Who should the bouncer run as (i.e., "nobody" or another specific user)?
- Error messages & explanatory text settings for various situations.

The address configuration file is relatively straight forward and allows the administrator to specify the old address, new address, some personal contact information for the recipient, and a configuration code used to determine which explanatory text is sent in the bounced messages for each user.

In an attempt to prevent mail loops, the bouncer follows RFC 822 [4] conventions for returning undeliverable mail. Additionally, in order to deal with mailing list programs which put the list maintainer's address in the Errors-To: header, it is given precedence. If there is no Error-To: header, the Sender: header is used to determine how to send the bounce to. In the absence of a Sender: header, the From: header will be used, and if the From: header does not exist, a last ditch effort is made using the sender's address from the SMTP message envelope.

Finally, any message with a Precedence: header matching an administrator-configurable setting will be handled by sending a separate not to the recipient, rather than replying to the sender. In this way, users will be reminded to update their mailing list subscriptions with their new address, and there is a much smaller risk of starting mail loops between the bouncer and a mailing list program. By default, Precedence: headers which trigger this behavior are any that match *bulk*, *junk*, *list* [5].

Logging Bounced Mail

Logging of all bounced mail is handled by sendmail itself, which will create an entry in the syslog output just as if the normal local delivery agent had been run. In this case however, the local delivery agent will be the bouncer, so it's relatively easy to pull a list of all bounced E-mail messages from the syslog output. A sample syslog entry is shown in Listing 5.

Sample Configuration

For a simple example of how the bouncer is configured, assume there is someone named John D Smith on one network, and another user named Jeff D Smith

```
R$+<@bouncer>  $#bouncer $:$1<@bouncer>      user@bouncer
```

Listing 3: Ruleset 0 rule to redirect bouncer messages

```
R$+<@$+.bouncer> $#bouncer $:$1<@$2.bouncer> user@*.domain.com.bouncer
```

Listing 4: Additional rule for *.bouncer

on a second network. John's E-mail address is *jdsmith@domain1.com* and Jeff's address is *jdsmith@domain2.com*. When the two networks are combined, it is desirable for each to answer for the new network (domain3.com) as well as both old networks (domain1.com and domain2.com) for backward compatibility. In this manner the same mail servers may be used to provide redundant delivery hubs for the same domain.

In order to implement this improved delivery system, one or both of the addresses above much change. Sendmail's *redirect* can't be used in this case because it only handles the case where one user changes their address; in this case, both users should change their addresses to avoid confusion, since different people may remember either Jeff or John as *jdsmith@[someplace].com*. If only Jeff were to change his userid, then John would likely start receiving mail in the future which the sender actually intended for Jeff, not realizing that Jeff had changed his userid.

The solution in this case is to change both addresses to something like *jdsmith1@newdomain.com* for John, and *jdsmith2@newdomain.com* for Jeff. In this way, both are unique and neither maintains the original address. Prior to changing the userid's, the bouncer is configured to respond to any of the following addresses:

```
jdsmith@domain1.com
jdsmith@domain2.com
jdsmith@newdomain.com
```

with a message that might look like that in Figure 1. This response would expedite the updating of address information that an external customer might be maintaining, while at the same time preventing phone calls to John or Jeff. It will also hopefully prevent the end users from having to contact a Network Helpdesk or the local postmaster to inquire why mail to jdsmith generates a 'username unknown...' message when it was successfully deliverable in the past.

The configuration of the bouncer program's reply is contained within a single ASCII file. The format of the file is:

```
code=user1,user2,[...]: \
    text (address1@[domain]), \
    text (address2@[domain]),[...]
```

where code is a two digit numeric code, user# is a username, text is any arbitrary text, and address# is the corresponding new address for user#. The domain is optional, and if left off, a default domain will be inserted automatically (configurable via an option in the bouncer code). By grouping multiple users on the same line or multiple lines using a backslash (\) as a line continuation character, the names and addresses displayed for a given address can be controlled. In the scenario for John and Jeff given above, the bouncer address configuration would look like:

```
01=jdsmith: John D Smith \
    (555-1212) (jdsmith1@), \
    Jeff D Smith (555-1234) \
    (jdsmith2@)
```

```
Sep 6 10:57:35 mailhub sendmail[527]: AA186621431: from=<user@somedomain.com>
Sep 6 10:57:35 mailhub sendmail[527]: AA186621431: size=80, class=0,
    pri=10080, nrcpts=2
Sep 6 10:57:35 mailhub sendmail[527]: AA186621431:
    msgid=<9708068735.AA873561403@somedomain.com>
Sep 6 10:57:35 mailhub sendmail[527]: AA186621431: relay=somehost [127.0.0.1]
Sep 6 10:57:36 mailhub sendmail[527]: AA186621431: to=jdsmith@bouncer,
    delay=00:00:01, stat=Sent, mailer=bouncer
```

Listing 5: Sample syslog entry.

```
Due to the consolidation of networks between domain1.com and
domain2.com to form a common network newdomain.com, some Electronic
Mail addresses were changed. This message is automatically generated
in response to your E-mail to one of the persons listed below. In
this case, we were unable to determine which user you intended to
send to, and request that you re-send your E-mail message to the
new address listed after the person's name below:

 John D Smith (Phone Number)    [jdsmith1@newdomain.com]
 Jeff D Smith (Phone Number)    [jdsmith2@newdomain.com]

For your convenience, a copy of your original message is included below.
======================================================================
[original message including headers]
```

Figure 1: Sample of a bounced message with indeterminate recipient.

Since no domain is specified, newdomain.com is appended automatically. This makes it simple to update if one of the jdsmith's moves into yet another domain later (i.e., leaves the company, changes departments, etc). It could then be updated to read:

```
01=jdsmith: John D Smith \
    (555-1212) (jdsmith1@), \
    Jeff D Smith (555-9875) \
    (jdsmith2@elsewhere.com)
```

The numeric code (01 in this example) is used by the bouncer program to determine which explanation is prepended to the bounced message. For example, you might set up the following codes and explanations:

- Code "01" to be used for a division merger with explanation of: Due to the consolidation of networks between domain1.com and domain2.com to form a common network newdomain.com, some Electronic Mail addresses were changed. This message is automatically generated in response to your E-mail to one of the persons listed below. In this case, we were unable to determine which user you intended to send to, and request that you re-send your E-mail message to the new address listed after the person's name below:
- Code "02" to be used for name changes with explanation of: The person you sent mail to has changed their mail address because their name changed. This message is automatically generated in response to your E-mail, and has been delivered to the new mail address for you. This is only an information message to allow you to update your records to reflect the user's new address.
- Code "03" may be used for users who have left the domain: The person you sent mail to listed below no longer has a mail address on this network. For security reasons, your message has not been forwarded to the person. Please re-send your message to their new address listed below if you still would like them to receive it.

A fully populated configuration file might then look like that shown in Figure 2. Once the bouncer is ready to handle the invalid addresses, the aliases file would be updated accordingly, as in Figure 3.

Alternative Solutions and Future

Alternative Solutions

As currently written, the bouncer behaves in a similar manner to the *vacation* [6] program. However, the bouncer is centrally managed and has finer control over what happens with each recipient's messages. The only configuration which needs to be done is to initially set the administrator options in the bouncer code, update the address information inside the single ASCII address configuration file, and configure the aliases for old usernames so they will be directed at the bouncer. If *vacation* were to be used for this implementation, each old account would need to be maintained, have a .forward file, and an individual *vacation* configuration. In the sample case provided earlier, *vacation* would not be a viable long-term solution when the password maps between domain1 and domain2 are eventually merged, because of the username conflict. While this problem can be circumvented with the creative use of aliases and sendmail.cf rules, the configuration of the bouncer is much simpler and obviates the need to maintain multiple configurations for each account that's changed.

Another alternative would be to take advantage of sendmail's *#error* delivery agent. However, this would require further modification of the sendmail.cf file in order to provide multiple error conditions, and tends to produce terse error messages. This would not meet the design goals of the bouncer – simple configuration and detailed explanations – and is more difficult

```
# Users who've had their address changed because of a namespace
# collision in the domain merger
01=jdsmith: John D Smith (555-1212) (jdsmith1@),\
    Jeff D Smith (555-1234) (jdsmith2@)

# Users who've had their address changed because of a name change
02=jadoe: Jane A Smith (formerly Jane A Doe, 555-1111) (jasmith@)

# Users who're no longer valid on this network
03=jsbrown: John S Brown (moved to Div XYZ) (jsbrown@xyz.domain.com)
```

Figure 2: Fully populated sample bouncer.cfg file.

```
jdsmith: jdsmith@bouncer        # can't tell who they want to send to here;
                                # just bounce it
jadoe: jasmith,jadoe@bouncer    # forward and notify sender of new address
jsbrown: jsbrown@bouncer        # dont fwd -- may have confidential
                                # information: just bounce it
```

Figure 3: Sample mail.aliases map for sample bouncer.cfg configuration.

to maintain, especially by junior administrators. The advantage to the bouncer is that once the sendmail.cf is initially set up, no further modifications need be made. Junior System Administrators can modify the bouncer.cfg and mail.aliases maps to configure new bouncer entries.

Future Direction

If enough traffic is being directed through the bouncer, it might be better implement in a compiled language such as C, to prevent the startup costs of Perl. However, the current delivery agent implementation takes less than 1 second to execute, so it would take a very large site to justify this more complex task. For example, our site contains over 10,000 accounts. If a namespace collision rate of 5% is assumed, this means that 500 usernames are going to be handled by the bouncer. If each address receives 10 pieces of E-mail per day, this is only 5,000 messages to handle. Most modern gateways can handle much more than this without any performance problems in an average day; ours typically process 75,000 to 100,000 messages in an average day with no noticeable performance problems. With the load balancing provided by MX record preferences [7], only extremely large sites may ever need to re-implement the bouncer for performance reasons.

The main feature lacking from the bouncer is the ability to auto-detect and stop mail loops. As much care as possible was invested in determining sender information and handling mailing list processors, but there is still small possibility that the bouncer program could get into a loop with another mail delivery agent such as a mailing list processor. While this shouldn't happen with such popular mailing list packages such as majordomo or listserv, not everyone uses these to process heir mailing lists, and not all mailing list processors follow the RFC's and use the correct header formats. An algorithm for detecting and preventing these types of mail loops will be added to future releases of the bouncer code.

Availability

Please contact the author directly for further information and program availability information.

Author Information

Rich Holland is the Technical Lead for the Enterprise E-mail Team at Rockwell Collins, Inc. where he is responsible for technical leadership and future direction in merging multiple mail systems into a common system for use by over 10,000 end users worldwide. Before coming to Rockwell, he was a Senior System Administrator for Synopsys where he oversaw the care and feeding of the Synopsys Porting Center machines. Reach him via U.S. Mail at Rockwell Collins, Inc.; M/S 106-193; 400 Collins Rd. NE; Cedar Rapids, IA 52498. His E-mail addresses are <rjhollan@collins.rockwell.com> and <holland@pobox.com>.

References

[1] *Sendmail 8.x (Berkeley) Release Notes*, ftp://ftp.cs.berkeley.edu/pub/sendmail.

[2] *Programming Perl*, Randal Schwartz and Larry Wall, O'Reilly & Associates, Inc., 1991.

[3] *Practical UNIX Security*, 2nd Ed., Simson Garfinkel and Gene Spafford, O'Reilly & Associates, Inc., April 1996.

[4] *RFC 822: Standard for the Format of ARPA Internet Text Messages*, Network Information Center, 1982.

[5] *Sendmail*, Brian Costales, Eric Allman, and Neil Rickert, O'Reilly & Associates, Inc., November 1993.

[6] *vacation*(1) man pages.

[7] *DNS and BIND*, 2nd ed., Paul Albitz, O'Reilly & Associates, Inc., January, 1997.

[8] Perl Home Page, http://www.perl.org/.

[9] AUSCERT Security Papers, http://www.auscert.org.au/information/papers.html.

USENIX ASSOCIATION

USENIX is the Advanced Computing Systems Association. Since 1975, it has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world.

USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems.

The USENIX Association and its members are dedicated to:
- problem-solving with a practical bias,
- fostering innovation and research that works,
- rapidly communicating results of research and innovation,
- providing a neutral forum for the exercise of critical thought and the airing of technical issues.

USENIX holds an annual multi-topic technical conference, the annual Systems Administration (LISA) conference, and frequent single-topic symposia addressing topics such as UNIX security, Tcl/Tk, object-oriented technologies, networking, electronic commerce, and operating systems design – as many as ten technical meetings every year. It publishes a magazine, *;login:*, eight times a year; and proceedings for each of its conferences and symposia. It also sponsors special technical groups as well as participating in various standards efforts such as IEEE, ANSI, and ISO.

SAGE, the System Administrators Guild

SAGE, a Special Technical Group within the USENIX Association, is dedicated to the recognition and advancement of system administration as a profession. To join SAGE, you must be a member of USENIX.

SAGE activities currently include publishing the "Short Topics in System Administration" series, the first three of which are "Job Descriptions for System Administrators," "A Guide to Developing Computing Policy Documents" and "Systems Security: A Management Perspective"; "SAGE News & Features," a regular section in *;login:*; The System Administrator Profile, an annual survey of system administrator salaries and responsibilities; co-sponsoring the LISA, SANS, and Large Installation System Administration of Windows NT conferences; support of working groups; encouraging the formation of local SAGE groups; and an archive site for papers from the LISA conferences and sysadmin-related documentation.

As a member of the USENIX Association/SAGE, you receive:
- Access to the papers from the USENIX Conference and Symposia proceedings, starting with 1993, via the USENIX Online Library on the World Wide Web (this includes the 1993, 1994, 1995, and 1996 LISA Conferences).
- Free subscription to *;login:*, the Association's magazine, published eight times a year, featuring technical articles, SAGE News & Features, columns on tools and techniques, book and software reviews, summaries of sessions at USENIX conferences, snitch Reports from various ANSI, IEEE, and ISO standards efforts, and much more.
- Discounts on registration for technical sessions at all USENIX conferences and symposia.
- Discounts on the purchase of proceedings and CD-ROMs from USENIX conferences and symposia.
- PGP Key signing service (available at conferences)
- Discount on the 4.4BSD Manuals plus CD-ROM published by the USENIX Association and O'Reilly & Associates, Inc.
- Special subscription rate to The Linux Journal.
- 20% discount of all titles from O'Reilly & Associates and Prentice Hall PTR.
- Savings (10-20%) on selected titles from McGraw-Hill, The MIT Press, Morgan Kaufmann Publishers, Sage Science Press, and John Wiley & Sons.
- Discount on all publications and software from Prime Time Freeware.
- Discount on software from BSDI, Inc.
- Right to vote on matters affecting the Association, its bylaws, election of its directors and officers.
- Right to join Special Technical Groups such as SAGE.
- SAGE members receive the following additional benefits: The most recent pamphlet in the "Short Topics in System Administration" series and the annual System Administrator Profile.

Supporting Members of the USENIX Association:

ANDATACO
Adobe Systems Inc.
Advanced Resources
Andrew Consortium
Apunix Computer Services
Boeing Commercial
Crosswind Technologies, Inc.
Digital Equipment Corporation
Earthlink Network, Inc.
ISG Technologies, Inc.
MTI Technology Corporation
Motorola Research & Development
O'Reilly & Associates
Open Market, Inc.
Sun Microsystems, Inc.
Tandem Computers, Inc.
UUNET Technologies, Inc.

SAGE Supporting Members:

Atlantic Systems Group
Digital Equipment Corporation
Enterprise Systems Management Corp.
Global Networking and Computing, Inc.
Great Circle Associates
OnLine Staffing
Pencom Systems Administration/PSA
Sprint Paranet
Taos Mountain
Texas Instruments, Inc
TransQuest Technologies, Inc.
UNIX Guru Universe

For more information about USENIX and SAGE membership, conferences, or publications, please contact:

The USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

Phone: +1 510 528-8649
Fax: +1 510 548-5738
Email: office@usenix.org
WWW: http://www.usenix.org

ISBN 1-880446-90-1